uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería en Tecnologías de la Telecomunicación
2018-2019

Trabajo Fin de Grado

"Diseño y desarrollo de una aplicación móvil de emergencias para situaciones de alerta para mujeres"

Laura Fernández Ávila Cobo

Tutora

Iria Manuela Estévez Ayres

Leganés, octubre 2019

Universidad Carlos III de Madrid

Escuela Politécnica Superior

Resumen

Se ha diseñado e implementado una aplicación para móviles con el sistema operativo Android, bautizada como ‘AppAlerta’. Está destinada a mujeres que se puedan encontrar ante la amenaza de una agresión o en cualquier situación de peligro.

La aplicación permite a la usuaria, de manera sencilla, emitir una señal de alerta mediante distintos medios de comunicación: llamada telefónica o mensaje de texto (SMS o Telegram). Además, posee otras funcionalidades como la grabación de audio, la geolocalización o la subida de estos datos a una cuenta de Drive, con el objetivo de que la hipotética situación de peligro quede totalmente registrada.

En definitiva, esta aplicación no solo tiene la intención de ayudar a la mujer en peligro a comunicar su situación de forma rápida para recibir la ayuda necesaria, sino que también pretende recolectar el mayor volumen de datos generados alrededor de dicha situación. Esta información podría ser de utilidad en una posible denuncia o investigación.

La aplicación ha sido desarrollada en el entorno de Android Studio, una herramienta sencilla para programar aplicaciones en Android.

El conocimiento previo de Java ha sido de gran utilidad para el desarrollo de esta aplicación. Se han adquirido competencias relacionadas con la tecnología Android, así como con la herramienta Android Studio. También ha sido necesaria la implementación de una base de datos en SQLite y de un bot de Telegram.

Por último, la aplicación utiliza numerosas APIs de Google, como las de Drive, Maps y Google Assistant, así como de la API de Telegram.

Agradecimientos

Como explico más adelante, he tenido muchas motivaciones para hacer el trabajo de fin de grado sobre este tema, pero la principal es que soy mujer, y me es muy difícil escuchar la cantidad de casos de violencia contra la mujer que se emiten cada día en las noticias. Por ello este trabajo se lo dedico a todas ellas, a todas las mujeres que han sufrido a causa de la violencia machista o incluso perdido la vida por ella. Ojalá este trabajo no fuera necesario.

Mis agradecimientos son para todas las mujeres de mi vida, mi madre y mi hermana las primeras, sin las que todo esto hubiera sido tremendamente difícil. También a mis amigas, las de siempre y las nuevas, que me han ayudado siempre que lo he necesitado. Finalmente, a mi tutora Iria, que aceptó supervisar este trabajo y me ha dado ideas y consejos siempre que lo he necesitado.

"Ignoramos nuestra verdadera estatura hasta que nos ponemos en pie"

Emily Dickinson

Contenido

Índice de Figuras

Índice de tablas

1.Introducción

En este capítulo se tratará de abarcar de una manera global los aspectos más generales del proyecto.

En primer lugar, se expondrán las motivaciones y causas que han llevado al diseño y desarrollo de AppAlerta. Asimismo, se enumerarán de forma cualitativa los objetivos principales de la aplicación, más adelante, en el capítulo de diseño, se detallarán los requisitos técnicos más específicos que deberá cumplir la aplicación. Se tratará también el análisis del marco regulador aplicable a este proyecto.

Finalmente, se incluirá un breve esquema de la organización de la presenta memoria.

1.1. Motivación

La idea que ha motivado el diseño y desarrollo de esta aplicación se cimenta en tres pilares bien diferenciados: el terreno didáctico, el social y el personal.

1.1.1. Motivaciones técnicas y didácticas

El grado de ingeniería en tecnologías de la telecomunicación es un grado muy general y por lo tanto abarca materias muy diversas. A la hora de elegir un tema para el TFG es imprescindible elegir a su vez una rama, la de telemática fue la elegida. Como alumna, mi intención era poder aprovechar el TFG para aplicar los conocimientos de programación adquiridos durante la carrera, pero también aprender otros nuevos. Aprender sobre un nuevo tipo de programación (Android y bases de datos) y su desarrollo en un entorno distinto a los utilizados durante el grado (Android Studio) fueron los motivos por los que decidí desarrollar una aplicación móvil en Android.

1.1.2. Motivaciones sociales

En cuanto al tema específico elegido para el diseño de esta aplicación, no puedo dejar de mencionar la terrible situación que sigue viviendo la figura de la mujer hoy en día. A pesar de ser una aplicación de alerta de funcionalidad genérica, su uso está destinado al colectivo de la mujer. Desgraciadamente, el índice de situaciones violentas, incómodas o dramáticas a las que se enfrenta este colectivo cada día, sigue siendo desorbitadamente alto.

Solo en lo que va de año, 38 mujeres han sido asesinadas a manos de sus parejas, como se muestra en la siguiente gráfica proporcionada por *epdata* (Recuento hasta el mes de agosto*) [1]*.

Fig. 1: Mujeres víctimas de violencia de género año 2019 [1].

Por otro lado, las denuncias por agresiones y delitos sexuales han crecido un 60% desde 2012, pasando de 6732 en 2012 a 10727 en 2018 [2]. Aunque estos datos resultan positivos puesto que significan que ha disminuido el miedo a denunciar y ha aumentado la conciencia social, se trata de un altísimo número de denuncias en un solo año.

Puesto que el objeto de este proyecto no es detallar ni analizar el aumento o la existencia de estas desgraciadas cifras, no se continuará dando datos, pero se cree que justifica debidamente el motivo para la elección de una aplicación de asistencia y ayuda en caso de alerta destinada a usuarias mujeres.

1.1.3. Motivaciones personales

Por último, mencionar que mi condición de mujer me hace vivir y empatizar más de cerca con los infinitos casos de violencia contra el colectivo que se escuchan en todos los medios diariamente. Con vistas a finalizar el grado y obtener el título de graduada en ingeniería, me gustaría reflejar con mi TFG lo que pretendo poder desarrollar a lo largo de mi carrera profesional: que la ingeniería tiene el objetivo de ayudar y facilitar la vida de las personas. Con esta aplicación pretendo poder aportar mi granito de arena en la lucha contra la violencia machista y el acoso sexual callejero.

1.2. Objetivos

El objetivo al que aspira AppAlerta es el de ser una aplicación de emergencia que las mujeres tengan instalada en su móvil y a la que puedan recurrir en caso de alerta.

La idea es conseguir una aplicación cómoda y sencilla mediante la cual, tras configurar una serie de preferencias, la usuaria pueda emitir una señal de alerta ya sea mediante una llamada telefónica, un SMS o un mensaje vía Telegram, a una serie de contactos que ella misma haya configurado como *contactos de emergencia.*

Además, la aplicación será capaz de activar el micrófono y grabar todo lo que esté sucediendo, guardando el archivo .mp3 en una carpeta dentro del dispositivo móvil.

También tendrá acceso a la localización del teléfono en todo momento, e irá asimismo escribiendo las coordenadas en un archivo de texto que será almacenado dentro de la misma carpeta.

Se ofrecerá como opción a la usuaria la posibilidad de subir a una cuenta de Google Drive ambos archivos en pequeños fragmentos que quedarán almacenados en la nube como prueba.

Todas estas acciones (las seleccionadas por la usuaria en el apartado de preferencias de la aplicación) se realizarán de forma simultánea pulsando el Botón del Pánico que ocupará casi la totalidad de la pantalla de inicio.

Por último, se establece como desafío conseguir la ejecución del Botón del Pánico utilizando un comando de voz por medio del asistente de Google.

En definitiva, el diseño y desarrollo de AppAlerta persigue los siguientes objetivos:

- Selección de un lenguaje de programación adecuado para el desarrollo de la aplicación.
- Elección de un entorno adecuado para el desarrollo de dicho lenguaje.
- Aprendizaje de dicho lenguaje, entorno o ambos.
- Desarrollo de una pequeña base de datos en la que almacenar los *contactos de emergencia* introducidos por la usuaria.
- Investigación e implementación de actividades en segundo plano en Android, así como su sincronización, para conseguir que todas las acciones deseadas se realicen en un solo *click*.
- Autonomía y control de la aplicación sobre todas las acciones, para que el usuario solo necesite de un *click*.
- Implementación de un bot de Telegram.
- Aprendizaje y uso de las APIs de Google y Telegram.

1.3. Marco Regulador

Como cualquier otra aplicación móvil, AppAlerta debe cumplir con el RGPD (Reglamento General de Protección de Datos) vigente en la Unión Europea desde el pasado mayo del 2018 [3].

Este reglamento busca proteger los datos personales de los usuarios e introducir transparencia en el uso que las aplicaciones hacen de estos datos. Este reglamento persigue principalmente cinco objetivos [3]:

- Seguimiento de datos: Será imprescindible mantener un seguimiento de los datos del usuario y e informarles de dónde están y qué se hace con sus datos.
- Seguridad: La aplicación debe garantizar la seguridad de todos los datos recolectados, sin importar su naturaleza.

- Aceptación del usuario: Será obligatorio contar con la aceptación del usuario de los términos y condiciones de la aplicación que informará de los datos recolectados, el porqué y la manera de usar, borrar, actualizar o explorar dicha información. La aplicación deberá solicitar el consentimiento del usuario cada vez que necesite acceder o hacer uso de sus datos.
- Derecho a borrar: El usuario deberá tener derecho a acceder a sus datos recolectados y solicitar, si así lo desea, el llamado *Derecho para Borrar* o *Derecho para ser Olvidado*, de manera que dicha información no pueda ser recuperada bajo ningún tipo de circunstancia.
- Extraterritorialidad: El reglamento RGPD también se aplicará a compañías no pertenecientes a la Unión Europea.

AppAlerta es una aplicación sencilla que no recolecta datos personales del usuario. Toda la información de la que hace uso se mantiene almacenada dentro del propio dispositivo y el usuario tiene pleno poder sobre ella. Tan solo cuenta con un servidor que ejecuta el bot y tiene acceso a los IDs de los chats de grupo de Telegram, como se explicará más adelante. Este servidor cuenta con los niveles de protección necesarios para garantizar la seguridad de los datos manejados.

Por otro lado, la aplicación realiza diversas acciones que necesitan del acceso a distintas funcionalidades del dispositivo, tales como la agenda de contactos, el micrófono, la localización, almacenamiento, llamadas, SMS, conexión a la red, etc. No obstante, esta aplicación ha sido desarrollada en Android, que ofrece sus propias medidas de seguridad. En versiones anteriores de Android, el usuario se descargaba la aplicación, y debía de leer y aceptar una larga lista de condiciones y permisos de acceso a distintas funcionalidades de su dispositivo para poder utilizar la aplicación. A partir de la versión 6 de Android, los permisos de clasifican en peligrosos y normales. Los de menor riesgo e imprescindibles para el funcionamiento de cualquier aplicación, se pedirán en el momento de la descarga, pero los peligrosos deberán pedirse en el momento en que la aplicación vaya a hacer uso de ellos. Una vez aceptados, la aplicación no deberá volver a pedirlos, pero el usuario podrá retirar su consentimiento en cualquier momento [4].

Por tanto, AppAlerta necesita de la aceptación de numerosos permisos clasificados como peligrosos, pero de acuerdo con la versión 6 de Android, los pedirá en el momento de ser utilizados, garantizando así la seguridad y la confianza del usuario.

1.4. Organización de memoria.

A continuación, se detallan cada una de las partes del presente documento con el fin de facilitar su lectura:

1. **Introducción:** Se han tratado las razones y motivaciones que han llevado a la autora del presente trabajo a su realización. También se han enumerado los objetivos que pretenden conseguirse a grandes rasgos. Por último, se ha expuesto el marco regulador que aplica a este proyecto y cómo se ha tenido en cuenta para el diseño e implementación de la aplicación.
2. **Estado del arte:** Incluirá un breve estudio de los trabajos ya existentes relacionados con este proyecto. Aplicaciones similares que hayan servido de motivación o inspiración. También se explorarán las diferentes tecnologías disponibles para el desarrollo de aplicaciones y bases de datos, y se justificará las elecciones tomadas para cada tecnología.
3. **Diseño:** Se expondrán los requisitos técnicos y el diseño funcional de la aplicación.
4. **Implementación:** Se relatará el trabajo realizado para lograr los objetivos anteriormente descritos.
5. **Resultados:** Se expondrán los resultados finales y la comparación con los objetivos iniciales.
6. **Evaluación:** Prueba de la aplicación sobre un grupo de usuarios y recogida de sus impresiones respecto al proyecto.
7. **Planificación y presupuesto:** Este capítulo pretende explicar de forma ordenada el proceso de realización de este proyecto, separando en bloques de trabajo bien diferenciados cada una de las tareas llevadas a cabo. También contará con una estimación del tiempo empleado para cada tarea.
8. **Conclusión y líneas futuras:** Finalmente, se cerrará el documento con una recopilación de conclusiones tanto objetivas como personales de los resultados obtenidos en este proyecto. También se propondrán mejoras y trabajo futuro.

2. Estado del Arte

2.1. Introducción

A continuación, se realizará un análisis de las diferentes tecnologías existentes relacionadas con el presente trabajo y se llevará a cabo una comparativa entre cada una de ellas. El estudio se realizará sobre los siguientes puntos:

- **Trabajos previos realizados:** Una recolección de algunas aplicaciones y proyectos relacionados con AppAlerta, con funciones similares u objetivos parecidos.
- **Aplicación:** Un breve resumen sobre las tecnologías disponibles para desarrollar una aplicación, tanto de la plataforma como de los entornos de desarrollo. Posteriormente se justificará debidamente los seleccionados para este proyecto.
- **Servicios de mensajería:** Una exposición sobre los distintos servicios de mensajería actuales, argumentando el elegido.
- **Almacenamiento:** Comparativa de las distintas formas de almacenar información y las elegidas para ser utilizadas en esta aplicación.

2.2 Trabajos previos relacionados

2.2.1. My112

La Comunidad de Madrid ha desarrollado una aplicación para móviles que realiza, en caso de emergencia, una llamada al 112 con geolocalización [5].

Fig. 2 Pantalla de inicio de My112 [5].

Fig. 3 Pantalla de llamada de emergencia de My112 [5].

Además, posee otras funcionalidades, permite la creación de una lista de contactos personales de emergencia a los que es posible enviar un SMS tras la llamada al Centro de Emergencias. También permite el envío de fotografías al 112 en caso de ser necesario el aporte de más información[5].

Asimismo, permite a los propios usuarios de la aplicación la suscripción a los centros integrados para recibir avisos en tiempo real de las emergencias reportadas. Creando así una red de conocimiento alrededor de las emergencias de cada zona[5].

Es válida en caso de situaciones de emergencia muy variadas, ya sea debido a fenómenos meteorológicos adversos, violentos o cualquier otra situación que pueda causar una alteración en la vida diaria de los ciudadanos. Una vez realizada la llamada, todos los dispositivos con My112 instalada que se encuentren dentro de la zona de peligro, recibirán una alerta [5].

Centros 112 integrados [5]:

- Centro 112 de Madrid
- Centro 112 de Castilla y León
- Centro 112 de Islas Baleares
- Centro 112 de Cataluña
- Centro 112 de Cantabria
- Centro 112 de Melilla
- Centro 112 de Navarra

2.2.2. S.O.S. Emergencias

Un equipo de Bomberos junto con un grupo de informáticos desarrolla este sistema de emergencias para servicios de bomberos. Se compone de cuatro aplicaciones conectadas entre sí: Central SOS, Navegador para vehículos, App profesional y App para ciudadanos [6]. Siendo esta última la que interesa abordar en este documento.

Fig. 4: Pantalla de inicio de S.O.S. Emergencias [6].

Fig. 5: Pantalla de servicios de S.O.S. Emergencias [6].

Su uso es similar al de My112, permite la realización de una llamada a los servicios de emergencias dada una situación de peligro. Solo permite el envío de ubicación a los servicios adheridos al sistema. No obstante, en el momento de realizar la llamada las coordenadas del usuario se mostrarán en la pantalla del dispositivo pudiendo indicar al operador su posición [6].

La aplicación está adaptada para personas con discapacidad auditiva o del habla por medio de pictogramas, así como para personas con discapacidad visual mediante el uso de Google Talkback [6].

S.O.S. está disponible en cuatro idiomas: inglés, español, francés y alemán [6].

2.2.3. Bindi

No es necesario irse muy lejos para encontrar más ejemplos de trabajos similares al presente. En la propia Universidad Carlos III de Madrid un equipo bautizado como UC3M4Safety está desarrollando desde 2016 un dispositivo para detectar y prevenir violencia contra las mujeres denominado Bindi [7].

En este proyecto están participando cuatro departamentos de la universidad (tecnología electrónica, telemática, teoría de la señal e ingeniería mecánica) junto con el instituto de estudios de género de la universidad [7].

El trabajo ha abarcado también trabajos de fin de grado, de fin de máster y doctorados [7].

El objetivo que persigue este equipo es desarrollar una solución tecnológica invisible que alerte en caso de detección de violencia o agresión hacia una mujer. Esta detección se realiza de forma automática por el dispositivo.
La tecnología consiste en la introducción de sensores en los conocidos *wearables* capaces de detectar las emociones de la víctima y de procesar el entorno auditivo, tanto la detección de la voz de la víctima como la del agresor, con el fin de reconocer si se trata de una situación de peligro [7].

Una vez el dispositivo emite la alerta, la idea es crear una red de ayuda alrededor de la víctima a través de las redes sociales o incluso de las fuerzas del estado para poder socorrerla de manera rápida y eficaz. Además, el dispositivo recopila toda la información posible de la situación con el fin de que esta pueda ser utilizada en un posible juicio [7].

Este increíble trabajo continúa avanzando hasta este año.

2.2.4. Conclusiones

Estos son solo algunos ejemplos en materia de tecnología destinada a situaciones de emergencia.

La última es increíblemente ambiciosa y por supuesto de ninguna manera se pretende conseguir algo similar en este proyecto, pero sí que se asemeja en el destinatario de esta tecnología, las mujeres, y en los objetivos principales que son la alerta, la asistencia y la recopilación de pruebas en caso de agresión, violación o acoso.

Las otras dos aplicaciones de emergencia, si bien son de carácter más general, también tienen un nexo común con este proyecto, y ese es el uso de un botón del pánico para alertar de una situación de emergencia. No obstante, AppAlerta pretende incluir algunas

funcionalidades extras y también persigue conseguir una aplicación más independiente del usuario, que sea capaz de realizar todas las acciones con un solo *click*. Mientras en estas dos aplicaciones el botón de emergencia solo abre el manejador de llamadas con el número de emergencia y es el usuario quien debe de pulsar una segunda vez para llamar, AppAlerta pide permisos al usuario para que todo esto se pueda realizar de forma automática, y ahorrar el máximo tiempo y responsabilidad al usuario.

Por otro lado, ambas aplicaciones ofrecen servicios destinados a contactar con organismos de emergencia. En AppAlerta la usuaria puede introducir como contacto de emergencia el 112, pero únicamente será de utilidad si configura como medio de contacto la llamada telefónica y ella misma comunica a los operadores su problema. No podrá, por el contrario, enviar un SMS o un mensaje de Telegram, AppAlerta no posee un acuerdo con ningún organismo oficial de emergencias para tal fin. Sería una mejora para el futuro de esta aplicación.

2.3. Aplicación

En este capítulo se explorarán las diferentes tecnologías existentes para la creación de una aplicación móvil.

2.3.1. Plataforma móvil

La elección de una plataforma móvil adecuada en el desarrollo de una aplicación es una de las decisiones más importantes a tener en cuenta para asegurar el éxito de la misma.

Hoy en día, existen cuatro plataformas principales, Android, iOS, Microsoft y Blackberry. Las principales características de cada una de ellas están detalladas en la Fig. 6 proporcionada por statista [8].

	Apple iOS 9 Apple iOS 9	Android 7.0 Android 7.0	Windows Phone 8 Windows Phone 8	BlackBerry 10 BlackBerry 10
Compañía	Apple	Open Handset Alliance	Microsoft	BlackBerry
Núcleo del SO	Mac OS X	Linux	Windows NT	QNX
Licencia de *software*	Propietaria	Libre y abierto	Propietaria	Propietaria
Año de lanzamiento	2007	2008	2010	1999
Fabricante único	Sí	No	No	Sí
Variedad de dispositivos	Modelo único	Muy alta	Media	Baja
Soporte memoria externa	No	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit/ Chromium (Blink)	Trident	WebKit
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry World
Número de aplicaciones*	2.400.000 (sept 2016)	2.000.000 (jun. 2016)	700.000 (oct. 2016)	270.000 (2016)
Coste publicar	$99 / año	$25 una vez	$99 / año	Sin coste
Otras tiendas sin supervisión	No	Si	No	Si
Familia CPU soportada	ARM	ARM, MIPS, x86	ARM	ARM
Soporte 64 bits	Si	Si	No	No
Máquina virtual	No	Dalvik / ART	.net	No
Lenguaje de programación	Objective-C, Swift	Java, C++	C#, Visual Basic, C++	C, C++, Java
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac
Multiusuario	No	Si	No	No
Modo invitado	Si	Si	No	No

Fig. 6: Comparativa de las principales plataformas móviles [8].

El primer factor a tener en cuenta a la hora de elegir una plataforma en la que desarrollar una aplicación para móviles es su cuota de mercado. En los últimos años, iOS y Android han quedado muy por delante del resto de aplicaciones.

La siguiente gráfica (fig. 7) muestra un estudio realizada por el *Grupo Gartner* que mide la evolución de las ventas de smartphone según el sistema operativo desde 2010 a 2018 [9].

En ella se observa como efectivamente, el mercado ha sido monopolizado por las plataformas de iOS y Android, relegando al resto de plataformas a su inminente desaparición.

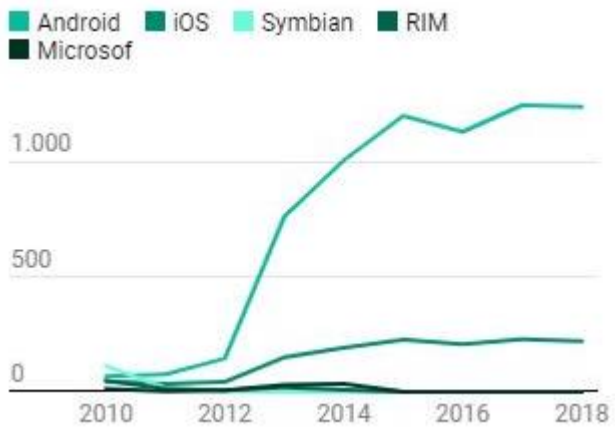


Fig. 7: Evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos desde 2010 a 2018. Unidades expresadas en miles de millones [9].

A continuación, se analizará qué diferencia a estas dos plataformas.

2.3.1.1. Android

En 2005 Android Inc., una pequeña compañía dedicada al desarrollo de aplicaciones móviles es comprada por Google. En 2007, se crea el consorcio Handset Alliance formado por múltiples compañías que tienen el objetivo de desarrollar estándares abiertos para móviles. Ese mismo año se lanza una primera versión de Android SDK. Al año siguiente se comercializa el primer móvil con Android (T-Mobile G1) y meses después se inaugura Android Market (más tarde conocido como Google Play), la tienda de descarga de aplicaciones en Android. Desde entonces, Android no ha hecho otra cosa que crecer y lanzar nuevas y mejoradas versiones, convirtiéndose en una plataforma líder a nivel mundial[10].

Android es una plataforma de desarrollo libre basada en Linux y de código abierto. Puede ser adaptada a cualquier tipo de hardware, no se limita a móviles y tabletas, la podemos encontrar en numerosos dispositivos como relojes o electrodomésticos. Esto facilita la escalabilidad y adaptación de una aplicación a distintos dispositivos, aunque el programador deberá tener en cuenta la diferencia de requisitos de memoria, pantalla, etc., de los distintos dispositivos [11].

Las aplicaciones finales son implementadas, por lo general, en el lenguaje Java, esto garantiza su portabilidad, ya que, por medio de una máquina virtual, podrían ser ejecutadas en cualquier tipo de CPU. También existe la posibilidad de utilizar Kotlin, un lenguaje más sencillo e intuitivo a la hora de programar o incluso en C# [12].

La seguridad la obtiene de Linux, los programas permanecen separados unos de otros gracias al concepto de ejecución dentro de una caja. Además, y como ya se ha mencionado antes en este documento en la sección del marco regulador, desde la versión 6.0 Android clasifica los permisos y obliga al desarrollador a solicitarlos en el momento de su uso. Estas características aportan seguridad y fiabilidad a Android [11].
Google Play es la tienda de compra de aplicaciones de Android. Las verificaciones que hace Android a la hora de aprobar la publicación de una aplicación son pobres en comparación con las que hace iOS. La publicación de una aplicación de Google Play tiene una tarifa única de 25 dólares [12].

2.3.1.2. iOS

iOS es una plataforma perteneciente a la multinacional Apple Inc. Originalmente se desarrolló en 2007 exclusivamente para el iPhone, con lo que el sistema operativo se denominó iPhone OS. El 6 de marzo de 2008 se liberó la primera versión beta de iPhone SDK y ya en 2010 se lanzó el iPhone OS. No es hasta la aparición del iPad, que utiliza el mismo sistema operativo, cuando pasa a llamarse iOS [13].

Apple desea tener el control total sobre su software y dispositivos por tanto iOS tan solo está disponible para iPhone y iPad. Además, a diferencia de Android, iOS no es adaptable, será necesario el desarrollo de una aplicación para iPhone y otra distinta para iPad [11].

Hasta 2014, el lenguaje utilizado para el desarrollo de aplicaciones móviles en Apple era Objective-C. Fue entonces cuando apareció Swift, un lenguaje más moderno, simplificado y sencillo [12].

La tienda de compra de aplicaciones, Apple App Store posee unos criterios de verificación ampliamente más estrictos que los de Google Play. Apple revisa sin excepciones cada aplicación que solicita su publicación en la tienda y puede llegar a tardar entre 1 y 2 días hasta dar su consentimiento [12].

Por otro lado, Apple App Store cobra una tarifa anual de 99 dólares por la publicación de una aplicación; considerablemente más alta que la de Google Play [12].

2.3.1.4. Conclusión

Realizando una comparativa de todo lo expuesto más arriba, se ha tomado la decisión de desarrollar una aplicación para móviles en la plataforma Android. Los principales motivos que han llevado a la toma de dicha decisión son los siguientes:

- Como ya se ha mencionado y como muestra la figura 7, la tasa de mercado de la plataforma Android está muy por encima de la de iOS. A la hora de desarrollar una aplicación se pretende que pueda llegar al máximo número de usuarios, y AppAlerta no es una excepción.
- Java es un lenguaje ya conocido por la desarrolladora de AppAlerta, por tanto, por motivos de eficiencia y agilidad en la programación, se ha optado por elegir la plataforma que utiliza este lenguaje, Android.
- Todas las pruebas que deberán realizarse de la aplicación se harán en un dispositivo Android. Aunque los entornos suelen ofrecer la posibilidad del uso de emuladores, para aplicaciones como esta, que usan localización, es más eficiente y efectivo el testeo en un dispositivo real.
- En cuanto a la adaptabilidad de Android a cualquier tipo de hardware, resulta extremadamente atractiva para esta aplicación, que en una línea futura podría desarrollarse para dispositivos wearables.
- Aunque la seguridad que ofrece iOS tanto del propio sistema operativo como de su tienda resulta más atractiva que la de Android, se cree suficiente para la aplicación que se quiere desarrollar.
- Tanto el desembolso inicial como a largo plazo resulta mucho menor para la plataforma Android que para iOS. Al tratarse de un proyecto aislado, sin financiación y joven, se considera Android la mejor opción.

2.3.2. Entorno de desarrollo integrado (IDE)

Un IDE no es otra cosa que un programa formado por una serie de herramientas de programación que permiten al programador desarrollar utilizando una interfaz más cómoda y agradable. La gran mayoría de ellos se componen de una serie de características básicas [11]:

- Editor de código
- Compilador
- Depurador
- Constructor de interfaz gráfica

Para el desarrollo de aplicaciones móviles Android, los más utilizados son Eclipse, Android Studio, Netbeans, Intellij y Aide. Se analizarán más en profundidad los tres primeros.

2.3.2.1. Eclipse

Eclipse es un entorno de desarrollo integrado, de código abierto y multiplataforma. Originalmente fue creado por IBM como sucesor de su conjunto de herramientas para VisualAge. Actualmente es desarrollado por la Fundación Eclipse, una fundación sin ánimo de lucro que persigue fomentar la idea de código abierto [15].

Fig. 8: Pantalla de inicio de Eclipse [14]

Es un IDE de código abierto y gratuito, considerado genérico ya que no pretendió ser utilizado mediante un único lenguaje de programación.

Aunque las funciones de Eclipse son típicamente generales, es posible ampliarlas mediante el uso de '*plugins'*, por medio de los cuales es posible añadir, por ejemplo, un control de versiones a través de *Subversion* y lograr una integración mediante *Hibernate [16]*.

Sus características principales son las siguientes [16]:

- Cuenta de un editor de texto con analizador sintáctico.
- Compilación en tiempo real.
- Tests unitarios con JUnit.
- Control de versiones con CVS.
- Integración con Ant.
- Asistentes (*wizards*) para la creación de proyectos, clases…
- Refactorización.

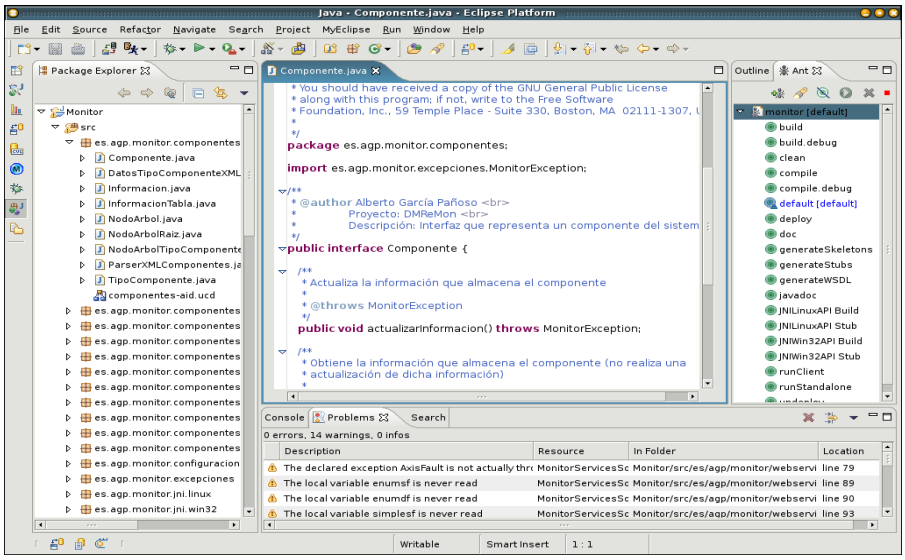


Fig. 9: Pantalla de trabajo del IDE Eclipse [16]

2.3.2.2. Android Studio

Android Studio fue anunciado el 16 de mayo de 2013 como sustituto de Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. Está basado en IntelliJ IDEA, del que posee un potente editor de códigos y sus herramientas de desarrollo [17].

Fig. 10: Logo de Android Studio [18].

Entre sus principales ventajas destacan [17]:

- Compilado rápido y flexible basado en *Gradle*.
- Emulador rápido y completo.
- Mismo entorno para el desarrollo de aplicaciones para toda la variedad de dispositivos Android.
- Ejecución inmediata para aplicar cambios sobre la aplicación en funcionamiento sin la necesidad de compilar nuevamente un APK.
- Integración con GitHub y otras plantillas de código para ayudar al desarrollador.
- Compatibilidad con C++ y NDK.

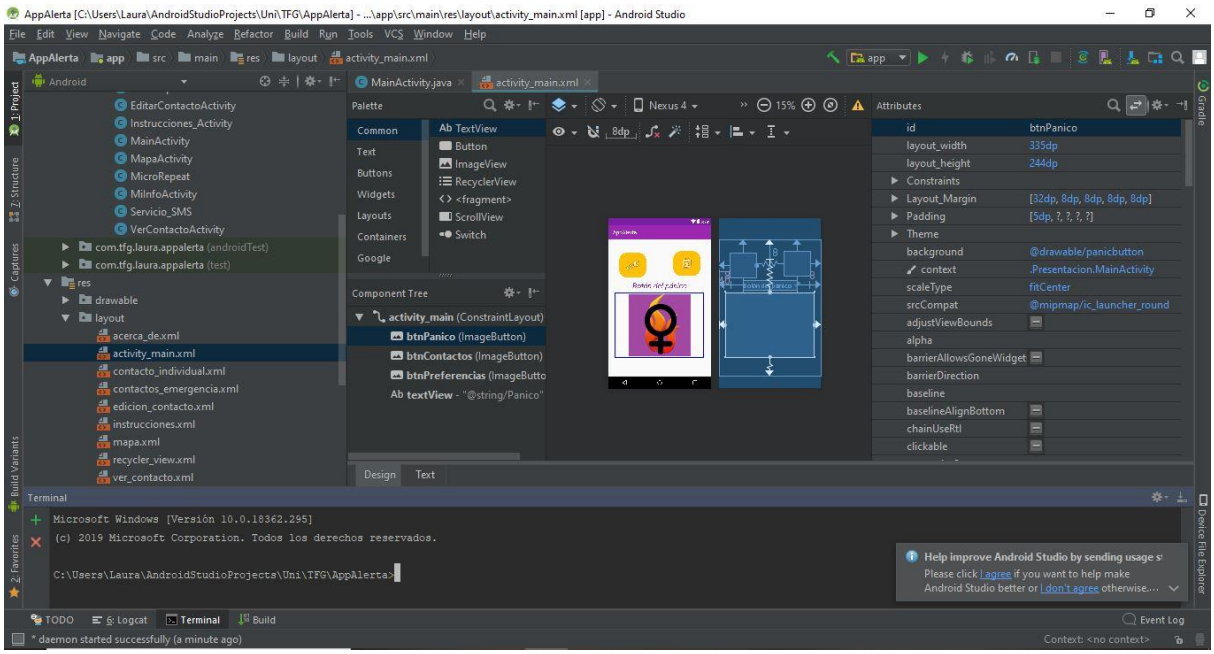


Fig. 11: Captura del IDE Android Studio del proyecto AppAlerta

2.3.2.3. NetBeans

Netbeans es un entorno de desarrollo integrado libre enfocado en un principio para el lenguaje Java. Es un proyecto de código abierto, gratuito y sin restricciones de uso fundado por Sun MicroSystems en junio del 2000 [19].

Su principal característica es que se basa en el concepto de módulos, componentes de software a partir de los cuáles las aplicaciones pueden ser desarrolladas. Estos módulos son archivos java que contienen clases de java que interactúan con las APIs de NetBeans y el archivo *manifest* [19].

Las aplicaciones desarrolladas por medio de módulos pueden ampliarse mediante la agregación de nuevos módulos, además, los módulos pueden desarrollarse de forma independiente lo que facilita la contribución o ampliación de las aplicaciones por distintos desarrolladores [20].

Fig. 12: Logo de IDE NetBeans [20].

NetBeans permite el desarrollo de todo tipo de aplicaciones Java y, al igual que Eclipse, cuenta con un sistema de proyectos basado en Ant, control de versiones y *refactoring* [19].

Posee un manejo de la memoria automático, utilizando el recolector de basura (*garbage collector*). No obstante, el tiempo de ejecución de las aplicaciones es algo más elevado que en el caso de Eclipse o Android Studio [21].

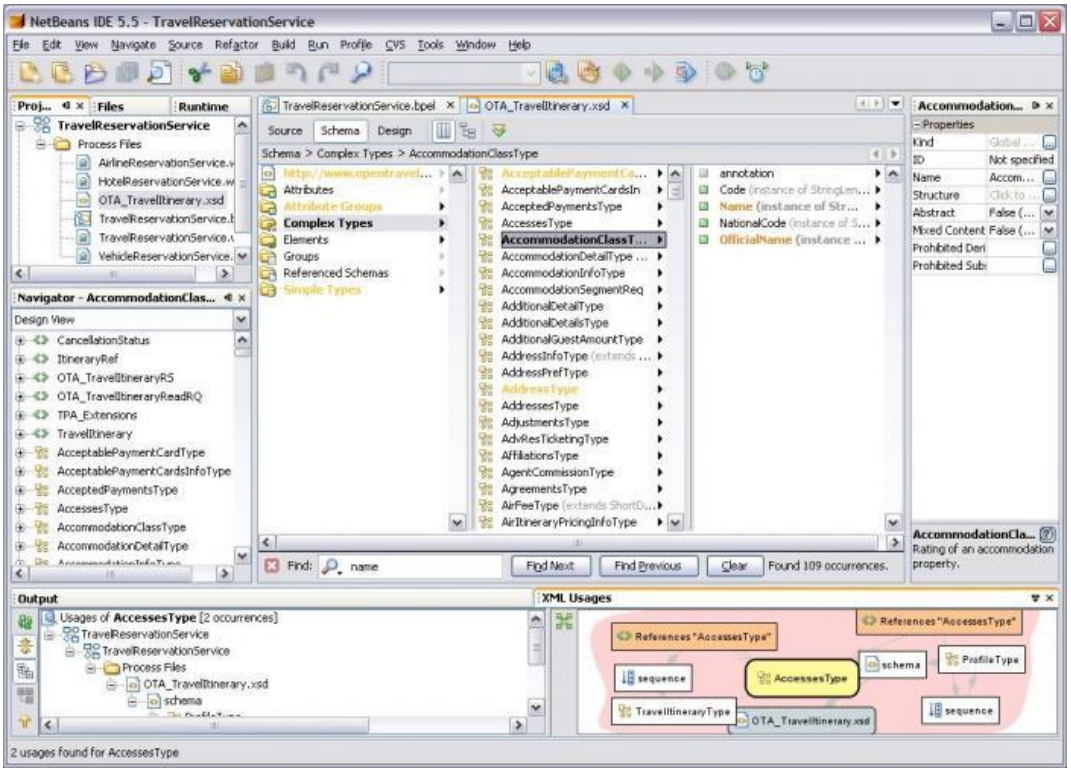


Fig. 13: Captura de entorno NetBeans [21].

2.3.2.4. Conclusión

Considerando estos tres entornos alternativos, se ha decidido utilizar Android Studio por diversos motivos.

Tanto Eclipse como NetBeans son entornos genéricos que permiten la creación de diversos tipos de aplicación. No obstante, Android Studio es el entorno oficial de desarrollo de aplicaciones móviles en Android, por tanto, se trata de un entorno especializado en la creación de este tipo de aplicaciones. Las interfaces son más intuitivas y orientadas a este tipo de desarrollo. Por ello, ya que este proyecto solo incluirá la creación y desarrollo de una aplicación móvil, se ha decidido utilizar el entorno oficial Android Studio.

2.3.3 Arquitecturas

2.3.3.1. MVC (Modelo-Vista-Controlador)

El patrón MVC es uno de los primeros modelos desarrollados en relación con el desarrollo software de interfaces gráficas de usuario. Fue creado por Trigve Reenskaug durante los años 70 [23].

Esta arquitectura divide el software de una aplicación en tres componentes o capas distintas[22]:

- **Modelo:** Esta capa se compone de todo lo relacionado con la representación de los datos del sistema, su lógica de negocio y sus mecanismos de persistencia. Es decir, contendrá aquellas funciones o métodos que tengan la labor de mostrar, actualizar, eliminar, etc. información de la base de datos.
- **Vista:** Consiste en la interfaz de usuario, eso es toda aquella información que se presenta al usuario.
- **Controlador:** Es la capa que hace de intermediario entre el Modelo y la Vista, maneja el flujo de información entre ambos componentes.

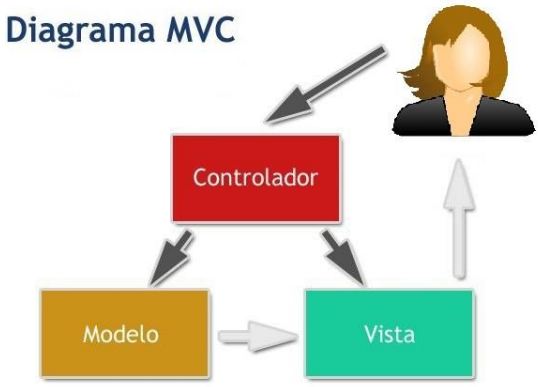


Fig. 14: Diagrama MVC (Modelo-Vista-Controlador) [23]

El objetivo de esta arquitectura es el de conseguir una estructura más clara y sencilla para el desarrollador. Mediante la separación de conceptos y la simplificación de cada parte es posible la reutilización de código. También facilita el mantenimiento de la aplicación[22].

2.3.3.2. MVP (Modelo-Vista-Presentador)

La arquitectura MVP, es un patrón basado en el MVC anteriormente mencionado que también divide la estructura del código en tres componentes [25]:

- Modelo: Capa que maneja los datos de la aplicación. También conocida como lógica de negocio.
- Vista: Capa que muestra los datos al usuario, en Android la constituirían las Vistas y los *Fragments*.
- Presentador: Capa que actúa de intermediario entre el modelo y la vista.

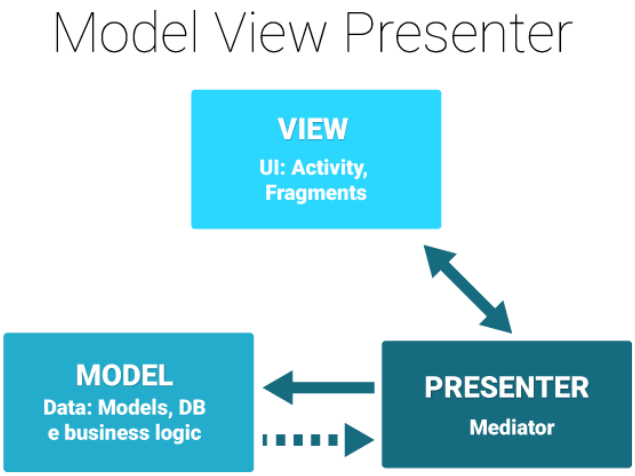


Fig. 15: Diagrama MVP (Modelo-Vista-Presentador) [24]

Es una estructura muy similar a la de MVC, no obstante, esta arquitectura libera de responsabilidad absoluta a la Vista, que no recibe ninguna información por parte del Modelo. Esto intensifica la independencia entre las capas y facilita las pruebas unitarias [24].

2.3.3.3. Arquitectura Clean

La arquitectura Clean fue desarrollada por el ingeniero Robert Cecil Martin, también conocido como Uncle Bob. Este ingeniero es también popular por ser el coautor de Manifiesto Ágil, así como de haber intervenido en el desarrollo de otros conceptos en ingeniería de software como los principios SOLID y TDD [27].

Esta arquitectura es una variante del patrón Hexagonal Architecture creado por Alistair Cockburn. El objetivo de esta arquitectura es dividir el software de un proyecto en capas

bien diferenciadas. Normalmente se utilizan cuatro capas, pero no hay una regla que defina este número como obligatorio [26]. La arquitectura se muestra en la figura 16.

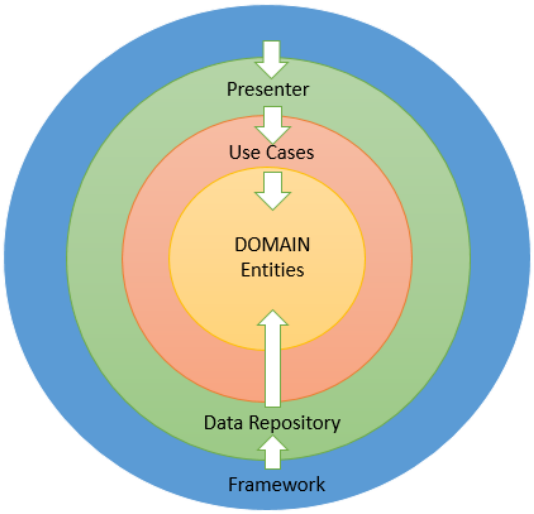


Fig. 16: Arquitectura Clean [27].

- **Capa Modelo (de Dominio o Lógica de Negocio).**
  Esta capa incluye todas las clases que representan la estructura interna del proyecto y la forma en que se representan los datos con los que se quiere trabajar.
- **Capa de Datos.**
  Incluye las clases destinadas a almacenar datos de forma permanente y que gestionan el acceso a estos datos, por ejemplo, bases de datos, servicios Web, preferencias, ficheros JSON, etc.
- **Capa de Casos de Uso.**
  Formada por el conjunto de clases que definen el conjunto de acciones que el usuario puede realizar mediante la aplicación.
- **Capa de Presentación.**
  Constituida por todas aquellas clases destinadas a la interacción directa con el usuario: actividades, fragments, vistas, etc.

  Como ya se ha mencionado, el uso de las cuatro capas no es obligatorio, pero sí se debe aplicar la *Regla de Dependencia*. Esta regla exige que todas las dependencias de código fuente deben apuntar siempre hacia dentro, por tanto, ninguna clase situada en la parte interior del círculo debe saber algo de las capas exteriores [26].

2.3.3.4. Conclusión

Tanto la arquitectura MVC como la MVP ofrecen una manera sencilla de organizar el código y permiten su reutilización, no obstante, se ha considerado que para el presente proyecto la arquitectura Clean es más adecuada. Su estructura de cuatro capas permite una mayor organización, y más flexibilidad a la hora de dar responsabilidad a cada Clase.

Esta arquitectura es además más adaptable a la estructura de un proyecto desarrollado en Android, ya que las actividades, aunque se considerarían vistas o interfaces con el usuario, también tienen peso en la programación. Por otro lado, la capa de Casos de Uso permite dividir las tareas a realizar por parte de la aplicación en paquetes más estructurados, manejables e independientes.

Por todo ello, la organización del código de este proyecto se basará en la arquitectura Clean. Más adelante, en el apartado destinado a esta arquitectura en la sección de implementación se enumerarán todas las clases y su organización (4.2.1.).

2.4. Servicios de mensajería

La principal función de esta aplicación es la de poner en contacto a la usuaria con otra persona para poder ser asistida en una situación de peligro.

Los medios por defecto para realizar esta comunicación son la llamada y el SMS, no obstante, actualmente la tecnología ofrece más posibilidades en lo que se refiere a mensajería que, aparte de ser servicios gratuitos, son ampliamente utilizados por la gran mayoría de usuarios con smartphone.

A continuación, se analizarán las principales alternativas y se concluirá con la elegida.

2.4.1. WhatsApp.

WhatsApp es un servicio de mensajería para smartphones que envía y recibe mensajes a través de la red. Comprende la mensajería instantánea, mensajes cortos e incluso multimedia (audio y vídeos). Permite, además, la creación de grupos o ‘chats’, llamadas y videollamadas. Todos estos servicios requieren evidentemente de conexión a Internet [28].

La aplicación fue adquirida por la empresa Facebook en 2014 por 19.000 millones de dólares cuando contaba con 450 millones de usuarios activos. Gracias a su aumento de popularidad y de la adición de nuevas funcionalidades, en 2018 ya superaba los 1.500 millones de usuarios mensuales con un intercambio de más de 60.000 millones de mensajes diarios [29].

2.4.2 Telegram.

Telegram es otra de las opciones de aplicación de mensajería que está ganando popularidad en los últimos años. Lanzada en 2013 y desarrollada por los hermanos Nikolái y Pável Dúrov, ofrece servicios similares a WhatsApp: mensajería de texto y multimedia o creación de grupos. Además, destacan otra serie de funcionalidades de las

que WhatsApp carece, como son los chats secretos cifrados de extremo a extremo, la posibilidad de envío de archivos de hasta 1.5GB, los canales de difusión o la creación y el uso de bots. Adicionalmente, permite el guardado de mensajes y la posibilidad de reenvío, alojamiento, sincronización y archivado desde la nube [30].

Contaba en 2018 con 200 millones de usuarios mensuales y sus creadores se jactan de la seguridad que ofrece esta aplicación frente a otros servicios de mensajería que han sido protagonistas de escándalos como el de Facebook y Cambridge Analytica [31].

2.4.3. Google Chat.

Google ha lanzado el pasado 2018 la nueva aplicación Chat, con la que pretende competir con el resto de las aplicaciones de mensajería [32].

Ofrece, al igual que WhatsApp o Telegram la posibilidad de crear grupos, enviar imágenes en alta calidad o *stickers* [32].

Debido a que sus funcionalidades son las básicas de cualquier aplicación de mensajería, Google ha decidido contar con el apoyo de las operadoras y basar su aplicación en el sistema RCS (Servicio de Comunicaciones Enriquecidas), nuevo estándar para los SMS. Funciona a través de la aplicación de mensajería del dispositivo, al enviar un mensaje a través de Chat si el receptor tiene un terminal compatible lo podrá recibir con el mismo formato, si no, lo recibirá como un SMS tradicional [32].

Con ello, Google pretende que Chat sea la evolución del SMS tradicional [32].

2.4.4. Conclusión.

Se desea que los medios de comunicación utilizados en AppAlerta sean rápidos, eficientes y de uso común con el fin de llegar a más usuarios.

La alternativa del Chat de Google queda descartada puesto que se trata de una aplicación poco popular que no ha llegado a demasiados usuarios todavía. Aunque los mensajes puedan ser enviados a un usuario sin dicha aplicación, los mensajes llegaran en formato SMS, y AppAlerta ya incluye la opción de envío de alerta a través de este mecanismo.

Por otro lado, Telegram ofrece una gran cantidad de servicios y una mayor seguridad, que lo hace un servicio atractivo. Además, Telegram dispone de una API pública, que permite a los desarrolladores la creación de aplicaciones que interactúen de manera sencilla con Telegram. Sin embargo, es cierto que todos estos servicios ‘adicionales’ no son realmente necesarios para AppAlerta y es cierto, que el número de usuarios que abarca es mucho menor que el de WhatsApp.

El servicio de WhatsApp es el más utilizado por los usuarios de manera que sería interesante para AppAlerta, ya que la usuaria tendrá la posibilidad de poder comunicarse con un mayor número de contactos. Sin embargo, no dispone de una API pública para desarrolladores, por tanto, toda la interacción entre WhatsApp y una aplicación externa ha de hacerse mediante *intents*. Esto supone que es imposible el envío de un mensaje a través de WhatsApp desde una aplicación externa de forma automática, es decir, es

posible abrir WhatsApp con el texto que se desea enviar, pero el usuario ha de pulsar en la tecla 'enviar' para que el mensaje sea expedido. Con ello, el servicio de WhatsApp queda descartado como tecnología de mensajería instantánea, el principal objetivo de este proyecto es que la usuaria pueda enviar el mensaje deseado con un solo *click* dentro de AppAlerta sin tener que entrar en la propia aplicación, si esta característica no se cumple, el objetivo principal de Applerta se pierde.

En conclusión, el servicio de mensajería elegido es Telegram.

Debido a distintas restricciones de Telegram, el mensaje de alerta a través de este medio de comunicación no puede hacerse de forma directa al chat de un contacto. No obstante, se ha encontrado una alternativa, el uso de bots de Telegram. El usuario crea un chat de grupo en el que incluye a sus contactos de emergencia y al bot creado para Telegram, 'AppAlerta'. La aplicación enviará los mensajes al grupo de alerta por medio de este bot. El bot recibirá peticiones de Telegram, y necesitará estar ejecutándose en un servidor.

Para las pruebas realizadas durante el desarrollo de esta aplicación, el bot ha sido ejecutado en el mismo PC en el que se ha desarrollado la aplicación,

En el punto 4.2.7., se analizará el funcionamiento de Telegram en mayor profundidad.

2.5. Almacenamiento

Para las alertas a través de llamada telefónica o SMS, AppAlerta requiere de la creación de una 'agenda' con contactos de emergencia de la usuaria. La aplicación recurrirá a esta agenda a la hora de realizar una llamada o enviar un SMS. Para que los datos de estos contactos se guarden en la aplicación y no se pierdan al reiniciar la misma, es necesaria la creación de una pequeña base de datos dentro de la propia aplicación, que almacene y gestione esta agenda.

Por otro lado, la principal característica de AppAlerta es el hecho de realizar una acción de llamada, SMS, Telegram, grabación de audio, etc., o todas, en un solo click y eso dependerá de lo que la usuaria quiere que se realice. Para ello, es necesario que exista una configuración inicial en la que la usuaria elige lo que quiere que ocurra cuando ella pulse el botón. Esa configuración deberá guardarse de alguna manera para que quede almacenada para una futura alerta.

Por último, la aplicación debe poder guardar la localización y los archivos de audio grabado para poder contar con los mismos como prueba.

A la hora de desarrollar una aplicación en Android existen distintos métodos para almacenar información de forma permanente [11]:

- **Bases de datos:** Las herramientas de Android dan soporte para SQLite, es posible la implementación de una base de datos en la aplicación de manera sencilla y con un alto rendimiento sin necesidad de servidor. SQLite es un modelo de base de datos SQL de código abierto, ligero y simple.

- **Ficheros:** Es posible almacenar la información que genera y utiliza una aplicación en ficheros dentro de la memoria del dispositivo o de una tarjeta SD.
- **XML:** Constituye un estándar en la representación de datos e información en Internet y en otros entornos como Android. Android facilita las librerías SAX y DOM para su utilización.
- **Preferencias:** Es una herramienta de Android que permite almacenar y acceder a datos primitivos del tipo pares clave/valor. Es útil para configuraciones iniciales de la aplicación.
- **JSON:** Otra opción para almacenar información estructurada utilizando una representación sencilla y compacta. Resulta útil para el intercambio de información a través de Internet.
- **Internet:** Siempre existe la posibilidad de almacenar o tomar datos de la nube.

Algunas de estas opciones han sido las elegidas para almacenar los datos de AppAlerta.

La agenda de contactos de emergencia se realizará por medio de una base de datos SQLite, ya que no necesita de una gran capacidad (solo serán algunos contactos de la usuaria) como para necesitar de un servidor, pero tampoco serán datos fijos y primitivos, como es el caso de las Preferencias. SQLite, permite crear una base de datos cómoda y rápida con una lista de contactos que pueda ser alterada o modificada en cualquier momento.

Las preferencias, sin embargo, resultan interesantes para la implementación de la configuración de opciones de la usuaria que se menciona más arriba. Una vez la usuaria decida lo que quiere que haga la aplicación, esta última solo deberá acceder a esta herramienta y pedir dicha información para desencadenar las acciones requeridas cuando se pulsa el botón del pánico.

Por último, los archivos de audio y localización pueden guardarse en ficheros dentro del propio dispositivo o ser subidos a la nube, a una cuenta de Google Drive. Ambas opciones tienen sus ventajas y desventajas. Guardar los archivos en el móvil requiere menos consumo de batería y no necesita de conexión a la red, no obstante, es cierto que requieren memoria en el teléfono, y en una situación de alerta, si el dispositivo se pierde, se deteriora o es robado los archivos guardados en su interior no serán de ninguna utilidad. Por otro lado, almacenar los archivos en la nube sí que supone un mayor consumo de batería y requiere conexión, pero su disponibilidad es total.

Debido a que se trata de una aplicación de uso ocasional, solo para emergencias, los datos que se almacenarían en el dispositivo serían puntuales, por ello se ha decidido implementar un sistema que por defecto guarde los datos en la memoria del teléfono móvil. Por el contrario, el gasto de batería se ha considerado más importante y únicamente si la usuaria lo elige, los datos serán subidos a Google Drive.

3. Diseño

3.1. Requisitos

3.1.1. Requisitos de la aplicación

Se entienden como requisitos de aplicación el conjunto de acciones que AppAlerta ofrece a la usuaria. La tabla 1 agrupa por categorías el total de funcionalidades que AppAlerta pretende conseguir:

Tabla 1 : Requisitos de aplicación

	Tipo	Requisito
1	Funcionalidades básicas	Configuración de preferencias.
2		Creación de una lista de contactos de emergencia y su gestión.
3		Acceso a contactos de la agenda del dispositivo para la creación de la lista
4		Creación de todas las vistas y pantallas de interacción con la usuaria.
5		Petición de permisos.
6		Administración de errores.
7		Sincronización de actividades.
8		Realización de una o todas las acciones únicamente mediante la pulsación del botón del pánico.
9		Elaboración de una versión en inglés de la aplicación.
10	Comunicación	Llamada a un contacto de la lista de forma automática.
11		Envío automático de SMS con texto predefinido por la usuaria a un contacto de la lista.
12		Envío automático de SMS con texto predefinido por la usuaria a todos los contactos de la lista.
13		Envío automático de SMS con ubicación a un contacto de la lista.
14		Envío automático de SMS con ubicación a todos los contactos de la lista.
15		Envío automático de SMS con texto predefinido por la usuaria y ubicación a un contacto de la lista.
16		Envío automático de SMS con texto predefinido por la usuaria y ubicación a todos los contactos de la lista.

17		Mensaje a través de la aplicación Telegram con texto, ubicación o ambos.
18	Localización	Acceso a la localización del teléfono.
19		Obtención de geolocalización del dispositivo.
20		Escritura de las coordenadas de la localización obtenida en un fichero de texto y su almacenaje en la memoria del terminal.
21		Escritura 'a trozos' de archivos de localización, para evitar perder el archivo completo si se interrumpe la conexión.
22	Micrófono	Activación del micrófono.
23		Almacenaje de ficheros de audio en la memoria del terminal.
24		Escritura 'a trozos' de archivos de audio, para evitar perder el archivo completo si se interrumpe la conexión.
25	Drive	Acceso a cuenta de Drive de la usuaria.
26		Subida de archivos de texto (localización) a Drive.
27		Subida de archivos de audio (micrófono) a Drive.
28	Google Assistant	Acceso al asistente de Google
29		Activación del botón del pánico por medio del asistente de Google al reconocer una palabra clave predefinida por la usuaria.

Tabla 1: Requisitos de aplicación.

3.2 Diseño de la aplicación.

A continuación, se detalla la manera en la que se han afrontado los ya enunciados requisitos de aplicación para su diseño.

3.2.1. Petición de permisos.

Como se ha expuesto más arriba en el Marco Regulador (1.3), a partir de la versión 6.0 de Android y de acuerdo con las últimas restricciones de seguridad en aplicaciones emitidas en el año 2018, Android desarrolla un modelo de petición de permisos. Clasifica los permisos en normales y peligrosos. Los normales, son permisos para realizar acciones no demasiado arriesgadas para el usuario, estos se piden al descargar la aplicación y han de ser aceptados por el usuario para que la aplicación pueda ser instalada. Los peligrosos, son permisos que comprometen los datos del usuario y Android fuerza a los desarrolladores a solicitarlos en el momento en el que la aplicación requiera de una acción que viole uno de esos permisos.

AppAlerta utiliza los siguientes permisos peligrosos:

- **CALL_PHONE:** Para realizar llamadas telefónicas de forma automática sin que el usuario tenga que presionar el botón de llamada.
- **SEND_SMS:** Para enviar mensajes SMS desde el dispositivo de forma automática, sin que el usuario tenga que pulsar la tecla enviar.
- **READ_EXTERNAL_STORAGE:** Para acceder a los archivos de audio y localización almacenados en el dispositivo y subirlos a Drive.
- **INTERNET:** Acceso a Internet.
- **READ_CONTACTS:** Para acceder a la agenda de contactos del dispositivo y que el usuario tome los que quiere que figuren en su lista de contactos de emergencia.
- **WRITE_EXTERNAL_STORAGE:** Para crear los archivos de audio y localización y almacenarlos en el dispositivo.
- **ACCESS_FINE_LOCATION:** Obtención de ubicación del teléfono.
- **RECORD_AUDIO:** Grabación de audio.

La aplicación deberá solicitar cada uno de estos permisos la primera que necesite acceder a esa determinada acción, una vez la usuaria haya aceptado el permiso, la aplicación no deberá volver a solicitarlo. En el apartado 4.2.4. se detalla la implementación de esta funcionalidad.

3.2.2. Configuración de preferencias.

Deberá definirse una lista de preferencias utilizando la herramienta de preferencias de Android mencionada en el punto 2.5. (Base de datos). A continuación, se adjunta la tabla 2 con las distintas configuraciones y su descripción.

Tabla 2: Configuración de preferencias

	Preferencia	Descripción
Información personal	**Nombre**	Información básica. Toma estos datos al enviar un mensaje para que el receptor sepa de quién es el mensaje de alerta en caso de no tener guardado su contacto.
	Apellido	
	Medios de comunicación	La usuaria elige los medios por los que quiere emitir la alerta: • Llamada • SMS • Telegram

SMS	**Cuántos SMS**	Permite elegir cuántos SMS se desean enviar. • Uno, solo al primer contacto de la lista de contactos de emergencia. • Uno a cada contacto de la lista.
	SMS con ubicación	Permite seleccionar el formato de SMS: • Únicamente un mensaje de texto de alerta. • Únicamente un mensaje con la ubicación de la usuaria. • Mensaje de texto y ubicación.
	Mensaje a enviar	La usuaria rellena con el mensaje que desea que se envíe en el SMS. Por defecto el mensaje será: "Hola, estoy en una situación de peligro y necesito ayuda".
Telegram	**Telegram ChatID**	ID del Chat de Telegram (Más información 4.2.7.).
	Telegram con ubicación	Permite seleccionar el formato de envío por Telegram: • Únicamente un mensaje de texto de alerta. • Únicamente un mensaje con la ubicación de la usuaria. • Mensaje de texto y ubicación.
	Mensaje a enviar	La usuaria rellena con el mensaje que desea que se envíe en el chat de Telegram. Por defecto el mensaje será: "Hola, estoy en una situación de peligro y necesito ayuda".
Micrófono	**Micrófono**	Permite activar el micrófono.
Drive	**Subir a Drive**	Ofrece la opción de subir a Drive los archivos: • Nada. • Archivo con grabación de audio. • Archivo con coordenadas de localización.
Comando de voz	**Comando por voz**	Permite la activación del botón del pánico por voz.
	Palabra clave	La usuaria puede introducir la 'palabra clave' que quiere utilizar para activar el botón del pánico. Por defecto la palabra clave es "ayuda".

Tabla 2: Configuración de preferencias.

3.2.3. Pantallas de la aplicación

AppAlerta se trata de una aplicación sencilla, cuyo objetivo es permitir al usuario, de forma clara y rápida, emitir una señal de alerta en un momento de pánico. Por ello, el diseño se ha entendido como una aplicación con pocas pantallas que contengan toda la información esencial de forma clara y concisa.

A continuación, se muestran las distintas pantallas y su función:

3.2.3.1. Pantalla principal

Constituye la pantalla de inicio y en ella se encuentran todas las acciones principales.

Fig. 17: Boceto Pantalla de inicio.

Contará con tres botones principales.

En la parte superior los botones de preferencias y contactos respectivamente. El botón de preferencias conducirá a la pantalla de configuración de preferencias. El de contactos a la pantalla de administración de contactos de emergencia.

En la parte inferior central y ocupando la mayor parte de la pantalla, el botón del pánico, botón fundamental de la aplicación, cuya pulsación desencadena todas las acciones preconfiguradas por el usuario en una sola acción.

En la barra de acciones se encontrarán:

- Un icono de ubicación para que el usuario pueda acceder a la interfaz de Google Maps y visualizar su localización en el mapa
- Un pequeño menú. Una pestaña de ‘Acerca de’ que explicará la función de AppAlerta, y otra de Instrucciones explicando el funcionamiento de esta.

3.2.3.2. Pantalla ‘Contactos de emergencia’

Una pantalla con *scroll* en la que figurará toda la lista de contactos seleccionados por la usuaria como contactos de emergencia. Abajo a la derecha se situará un botón ‘+’ para añadir contactos a la lista que redirigirá a la usuaria a la pantalla de *Añadir-Editar Contacto*. Cada contacto de la lista podrá ser seleccionado de forma individual para visualizar por separado y realizar otras acciones en la pantalla de *Ver Contacto*.

A continuación, se ofrece un boceto de la pantalla (Fig. 18):

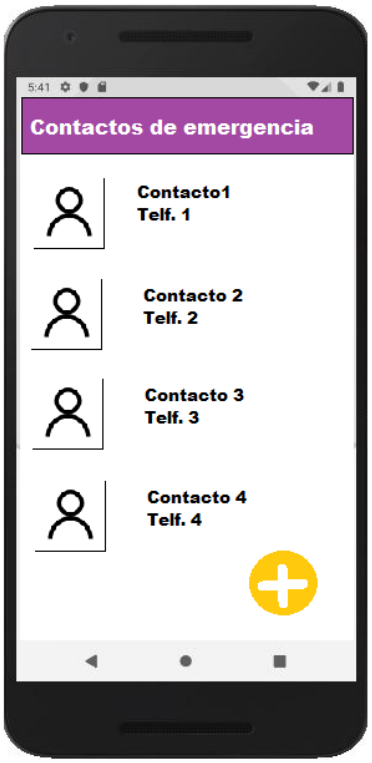


Fig. 18: Boceto Pantalla de Contactos de Emergencia.

3.2.3.3. Pantalla ‘Ver Contacto’

Pantalla a la que se accede pulsando en cada uno de los contactos de la lista de la pantalla de C*ontactos de Emergencia*. Mostrará los datos del contacto seleccionado. En la barra de tareas contará con un pequeño menú con las opciones de Borrado y Editado de dicho contacto.

El botón de borrado mostrará un mensaje de confirmación en pantalla para borrar el contacto. Si la usuaria pulsa aceptar el contacto se eliminará de la base de datos, automáticamente se volverá a la pantalla de C*ontactos de Emergencia* actualizada.

El botón Editar redirigirá a la usuaria a la pantalla de *Añadir-Editar Contacto*. La figura 19 muestra un breve boceto de la pantalla.

Fig. 19: Boceto Pantalla Ver Contacto

3.2.3.4. Pantalla 'Editar contacto'

Pantalla a la que se accede bien pulsando el botón editar en la pantalla *Ver Contacto*, o pulsando el botón '+' de añadir contacto en la pantalla de *Contactos de Emergencia*.

Contará con dos cuadros de texto a rellenar para el nombre y el teléfono de contacto. Si se ha accedido a la pantalla para editar un contacto, los cuadros de texto aparecerán completados con el nombre y teléfono actuales, si, por el contrario, se pretende añadir un nuevo contacto a la lista, los cuadros aparecerán vacíos.

Se permitirá añadir o editar un contacto de forma manual, escribiendo directamente sobre los cuadros de texto, o buscándolos en la agenda del teléfono. Para esta última acción se requerirá pulsar sobre el botón BUSCAR de la derecha. Este botón abrirá la agenda de contactos y la usuaria podrá seleccionar el que desea añadir. Una vez seleccionado, se volverá a la pantalla actual y los cuadros de texto estarán completados con los datos del usuario seleccionado en la agenda.

Por último, en la barra de tareas se visualizará un menú con las opciones de Cancelar y Guardar.

En caso de pulsar Cancelar, no se realizará ninguna acción y se regresará a la pantalla de *Contactos de Emergencia* si se pretendía añadir un nuevo contacto, o a la de *Ver Contacto* si se pretendía editarlo.

Si se pulsa el botón de Guardar, el contacto nuevo o editado se guardará en la base de datos y se retrocederá a la pantalla de *Contactos de Emergencia* actualizada si se trata de un nuevo contacto, o a la pantalla de *Ver Contacto* actualizada si se ha realizado una modificación.

A continuación, se muestra un boceto de esta pantalla (Fig. 20).

Fig. 20: Boceto Pantalla Añadir-Editar Contacto.

3.2.3.5. Otras pantallas.

Por último, la aplicación contará con tres pantallas más.

La pantalla de *Configuración de Preferencias* elaborada con la herramienta de preferencias de Android. En esta pantalla se visualizarán todos los campos mostrados en la tabla 1. La usuaria deberá elegir entre las opciones el tipo de configuración que prefiere para el momento de pulsar el botón del pánico. Después la aplicación podrá acceder a estos datos haciendo uso de la librería de preferencias. Más adelante, en el capítulo de implementación (4.2.3.), se expondrá en mayor detalle cómo se realiza el acceso a estos datos.

La pantalla de *Acerca de…* situada en el menú de la barra de tareas de la Pantalla de Inicio. En esta pantalla se leerá un breve texto explicativo contando en qué consiste la aplicación.

Finalmente, la pantalla de *Instrucciones.* Como ya se ha mencionado anteriormente, el objetivo de esta aplicación es que cualquier usuaria pueda hacer uso de ella, sin importar su nivel de conocimientos técnicos, por ello se ha decidido incluir una pantalla en la que explicar en detalle y de manera visual, los pasos que se han de seguir para poner en funcionamiento la aplicación. También se dedicará una breve sección ilustrativa al uso de esta aplicación con Telegram, ya que necesita de una explicación más detallada. En el capítulo de implementación de este documento se analizará este punto de forma específica (4.2.7.).

3.2.4. Administración de errores

A continuación, se recogen algunos de los mensajes de error que deberán aparecer a la hora de intentar realizar determinadas acciones:

- Error en la pantalla de *Editar-Añadir Contacto* al intentar añadir o guardar un contacto repetido en la lista.
- Mensaje de error al pulsar el botón del pánico, en caso de no haber seleccionado ningún medio de comunicación en el apartado 'medios' de la pantalla de *Preferencias*.
- Error al pulsar el botón del pánico si en las preferencias la usuaria ha seleccionado que desea subir a Drive el audio grabado, pero no ha marcado la casilla de grabación de audio.
- Mensaje de error al pulsar el botón del pánico si en la pantalla de preferencias se han seleccionado los medios Llamada, SMS o ambos, y la usuaria no ha incluido ningún contacto de emergencia en la lista.

3.2.5. Sincronización de actividades.

Como ya se ha mencionado, el atractivo de esta aplicación es conseguir que, con un solo gesto, pulsando un botón o gritando una palabra clave, se desencadenen una serie de acciones, todas al mismo tiempo.

Normalmente, en los lenguajes de programación utilizados, las funciones o métodos se ejecutan de manera sucesiva, no se comienza el siguiente método hasta que no ha terminado el anterior. Para conseguir que las acciones se realicen de forma paralela es necesario implementar algún tipo de mecanismo.

En Android se utilizan las actividades en segundo plano y los servicios. En esta aplicación necesitamos que hasta un total de seis acciones se realicen a un mismo tiempo: llamada, SMS, mensaje por Telegram, grabación y escritura de archivo de audio, obtención de localización del dispositivo y su escritura en un archivo, subida de archivos a Drive. Para

ello será imprescindible el uso de servicios y actividades en segundo plano. En el apartado 4.2.6., se explican con más detalle estos mecanismos.

3.2.6. Opción en inglés.

Resulta interesante que una aplicación pueda visualizarse en más de un idioma, de esta forma es posible llegar a un público más amplio. Se quiere que AppAlerta tenga una versión en inglés. Todos los textos explicativos, de error, petición de permisos, botones, etc., aparecerán en inglés si el idioma configurado en el dispositivo es el inglés.

3.3. Diseño de la base de datos

El objeto principal de AppAlerta es '*Contacto*'. Sobre este objeto se realizan todas las operaciones: añadir nuevo, editar, llamar, enviar mensaje, borrar, etc. Para poder realizar toda esta serie de operaciones sobre el mismo, será necesario almacenar las distintas características de las instancias de este objeto en una base de datos.

La base de datos como ya se ha mencionado, se realizará mediante SQLite. Para ello se creará la tabla '*contactos*' en la que se almacenará la siguiente información para cada uno de los contactos:

- **_id:** entero utilizado como clave principal de la tabla contactos.
- **Nombre:** nombre del contacto.
- **Estado:** estado del contacto.
- **Contacto_id:** id único del contacto tomado del ContentProvider de la agenda del teléfono.
- **Fecha:** fecha de creación del contacto en AppAlerta.
- **Teléfono:** número de teléfono del contacto.

Más adelante, en el punto 4.3., se analizará el funcionamiento de esta tabla.

4. Implementación

A continuación, finalizado el diseño de las distintas partes del proyecto, se analizará en detalle el proceso de creación e implementación de los componentes de AppAlerta.

4.1. Medios utilizados

Para el desarrollo de ambos, bot y aplicación, se ha utilizado un ordenador portátil con las siguientes características:

- Procesador Intel Core i7 6500U, 2.59 GHz
- Memoria 8GB
- Disco duro SSD de 250 GB
- Sistema Operativo: Windows 10 Home

El entorno de desarrollo empleado ha sido Android Studio 3.1.

Para la elaboración de esta memoria se ha utilizado el editor de texto Microsoft Word 2016.

4.2. Implementación de la aplicación

4.2.1. Arquitectura Clean

Como ya se ha mencionado en el apartado de estado del arte (2.3.3.2.), el código de este proyecto se ha estructurado siguiendo la arquitectura Clean. Según esta arquitectura, el conjunto de clases, interfaces, servicios, etc. ha quedado organizado de la siguiente manera (ver figura 21):

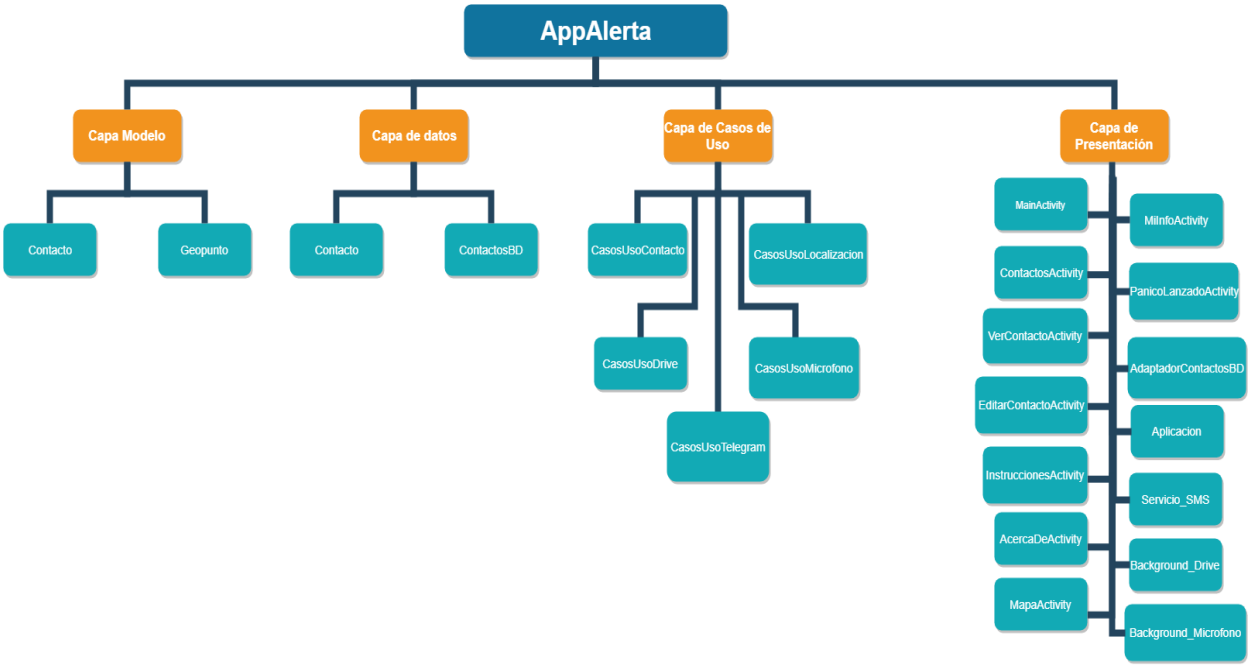


Fig. 21: Arquitectura Clean AppAlerta

- **Capa Modelo**
    1. **Contacto:** clase que define el objeto Contacto, conformado por los siguientes atributos:
        - Nombre.
        - Estado.
        - Contacto_id.
        - Teléfono.
        - Fecha.
    2. **GeoPunto:** clase que define el objeto GeoPunto, utilizado para la obtención de la localización del dispositivo. Sus atributos son únicamente las coordenadas.
        - Latitud.
        - Longitud.

- **Capa de Datos**
    1. **Contactos:** interfaz que define las distintas operaciones disponibles a realizar sobre una lista de contactos. Estas acciones son las siguientes:
        - *Contacto contacto (int id)* → Devuelve el objeto contacto dado su _id, clave principal de la tabla contacto.
        - *int nuevo()* → Añade un contacto y devuelve su _id.
        - *void borrar(int id)* → Elimina el contacto con dicho id.
        - *int size()* →Devuelve el número de contactos almacenados.
        - *void actualiza (int id, Contacto contacto)* → Realiza una modificación sobre el contacto con dicho id.
        - *boolean BuscaContactoId(String id)* → Busca el contacto con el 'contacto_id' aportado.
        - *boolean BuscaContactoTelf(String telf)* → Busca el contacto con el teléfono indicado.

    2. **ContactosBD:** clase que crea y administra la base de datos de contactos, implementando la interfaz Contactos. Desarrolla los métodos de la interfaz contactos y además incluye:
        - *Void onCreate(SQLiteDatabase bd)* → Crea la tabla contactos.
        - *Contacto obtieneContacto(Cursor cursor)* → Crea un nuevo contacto con los datos de la posición actual de un cursor.
        - *Cursor extraeCursor()* → Devuelve un cursor con todos los datos de la tabla.

    El punto 4.3., trata la implementación de esta base de datos.

- **Capa de Casos de Uso**
    1. **CasosUsoContacto:** clase que recoge todas las acciones realizadas sobre el objeto Contacto.

2. **CasosUsoDrive:** clase que implementa las APIs de Drive y realiza todos los métodos relacionado con subida de archivos a Drive.
3. **CasosUsoLocalizacion:** clase que realiza todas las acciones destinadas a obtener y manejar la localización del dispositivo.
4. **CasosUsoMicrofono:** clase que implementa todos los métodos relacionados con la activación del micrófono y su almacenamiento en forma de archivo en el dispositivo.
5. **CasosUsoTelegram:** clase que administra los mensajes a través de Telegram.

- **Capa de Presentación**
    1. **MainActivity:** pantalla principal de AppAlerta en la que se realizan las principales acciones de la aplicación.
    2. **ContactosActivity:** pantalla de Lista de Contactos.
    3. **VerContactoActivity:** pantalla que visualiza un único contacto.
    4. **EditarContactoActivity:** pantalla para adición o edición de un contacto.
    5. **InstruccionesActivity:** pantalla de instrucciones.
    6. **AcercaDeActivity:** pantalla con explicación breve.
    7. **MapaActivity:** pantalla que carga el mapa de GoogleMaps con la posición actual.
    8. **MiInfoActivity:** pantalla de configuración de preferencias.
    9. **PanicoLanzadoActivity:** pantalla que aparece al pulsar el botón del Pánico.
    10. **AdaptadorContactosBD:** clase que implementa en forma de RecyclerView la vista de la actividad ContactosActivity.
    11. **Aplicación:** clase genérica que permite acceder a distintos objetos desde cualquier parte del proyecto.
    12. **Servicio_SMS :** servicio que se ejecuta en segundo plano e implementa el envío de SMS.
    13. **Background_Drive:** actividad en segundo plano que ejecuta las acciones de CasosUsoDrive.
    14. **Background_Microfono:** actividad en segundo plano que ejecuta las acciones de CasosUsoMicrófono.
    15. **DriveRepeat:** Clase que hereda TimerTask para ejecutar repetidamente las acciones en Drive.
    16. **MicroRepeat:** Clase que hereda TimerTask para ejecutar repetidamente la grabación.

La figura 22 muestra un diagrama que ilustra la relación entre cada una de estas clases.

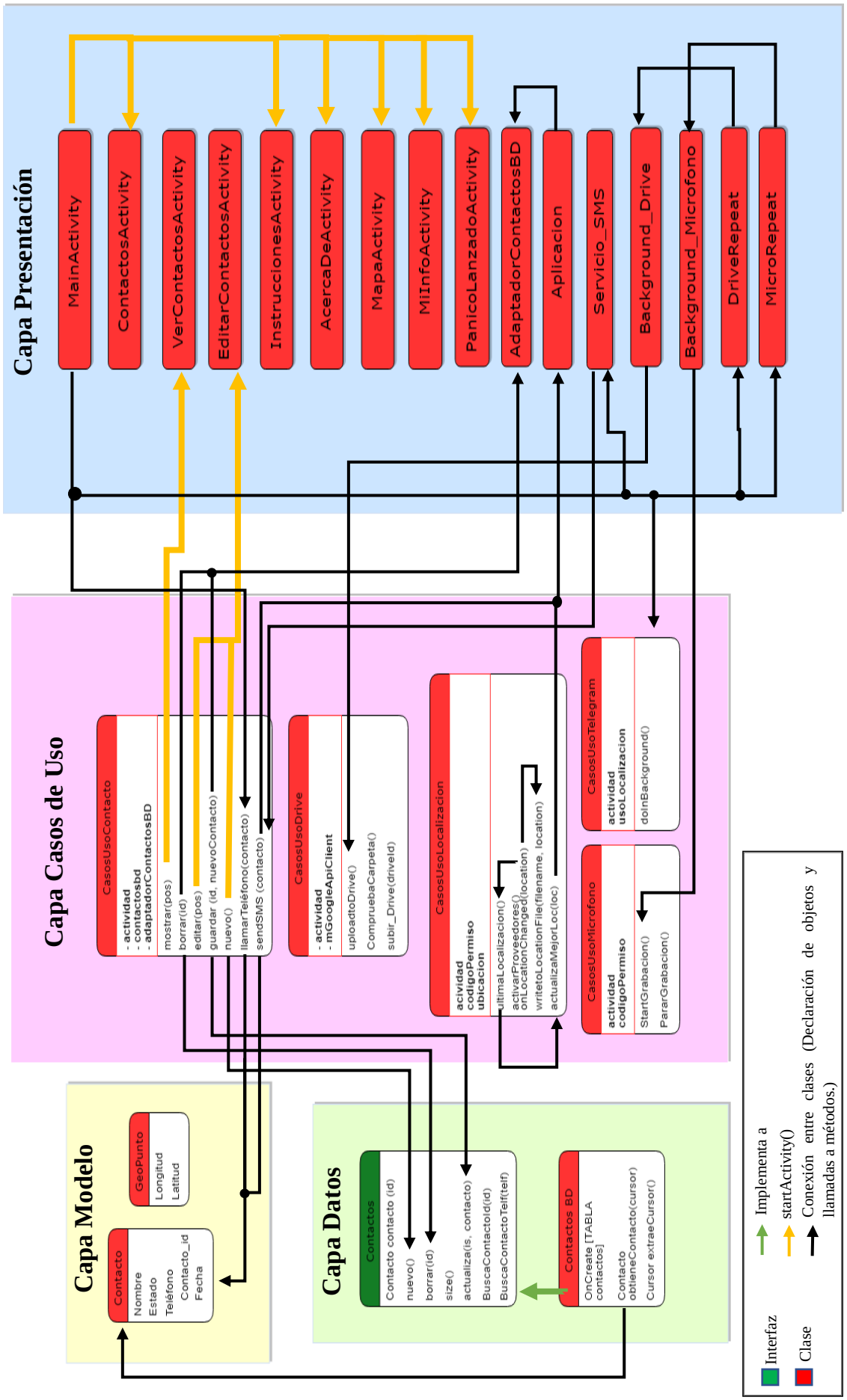


Fig. 22: Relación entre clases.

4.2.2. Casos de Uso

A continuación, se analizan cada uno de los casos de uso expuestos con anterioridad, explicando los objetos que se definen en ellos y las acciones que se realizan sobre los mismos.

4.2.2.1. Capa Casos de Uso Contacto

Esta clase administra todas las operaciones relacionadas con el objeto Contacto. Su constructor recibe la actividad desde la que se llama a cada método, la base de datos contactos, y el adaptador contactos que, en definitiva, es la base de datos que se visualiza por medio del *RecyclerView*.

Esta clase recoge los métodos básicos de administración de contactos:

- **Mostrar:** método llamado al pinchar sobre un contacto de la pantalla en la que se visualiza la lista de contactos de emergencia. Inicia la actividad VerConstactoActivity que mostrará el contacto en esa posición.
- **Borrar:** elimina el contacto con el id dado, borrándolo de la tabla contactos.
- **Editar:** Inicia la actividad EditarContactoActivity para el contacto en esa posición.
- **Guardar:** guarda el contacto en la tabla mediante el método actualiza().
- **Nuevo:** llama al método nuevo() de ContactosBD para obtener el nuevo id de contacto e inicia la actividad EditarContactoActivity.

Después, incluye los métodos específicos:

- **Llamar por teléfono:** recibe un objeto de tipo Contacto del que obtiene el número de teléfono a quién se realizará la llamada. Después de comprobar los permisos, inicia la intención ACTION_CALL que iniciará la llamada de forma automática.
- **Enviar SMS:** recibe también el contacto del que extrae el teléfono. Por otro lado, examina las preferencias para ver cuántos SMS quiere enviar la usuaria, si quiere que sean con o sin ubicación, y qué texto desea que se envíe. Después, y tras comprobar los permisos, utiliza el Manejador de SMS de Android para enviar el mensaje.

```
try {
    SmsManager.getDefault().sendTextMessage(numeroTel2, null,
messageUbic, null, null);

    } catch (Exception e) {
     e.printStackTrace();
    }
```

4.2.2.2. Capa Casos de Uso Micrófono

Esta clase maneja la grabación de audio y su escritura en un fichero dentro del dispositivo. Hereda la clase MediaRecorder de Android. Su constructor recibe como parámetros la actividad y el código de permiso.

Desde cualquier actividad en la que se instancia un objeto de tipo CasosUsoMicrofono se puede llamar a su método StartGrabacion().

Este método primero comprueba que hay permiso de grabación (RECORD_AUDIO) y de almacenamiento (WRITE_EXTERNAL_STORAGE) y a continuación prepara la grabación.

Primero genera un *String* con una secuencia de números aleatorios para nombrar cada fragmento de audio. Estos números los guarda en orden en una lista para poder acceder más tarde a ellos a la hora de subirlos a Drive. Asocia un flag a cada uno de ellos que pone a 0 para indicar que todavía no han sido subidos a Drive.

Después, genera el fichero dentro de la carpeta AppAlerta con la forma AudioRecording_nºRandom.mp3.

Finalmente, crea un objeto de tipo MediaRecorder al que le da formato y otras características y asocia al fichero mencionado. La grabación comienza cuando se llama al método start() desde el objeto creado.

```
Random = UUID.randomUUID().toString();
RandomListAudio.add(Random);
flag_audio = 0;
FlagListAudio.add(flag_audio);

mFileName =
Environment.getExternalStorageDirectory().getAbsolutePath() +
"/AppAlerta/AudioRecording_" + Random + ".mp3";

mRecorder = new MediaRecorder();
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
mRecorder.setOutputFile(mFileName);
try {
    mRecorder.prepare();
    mRecorder.start();
} catch (IOException e) {
    Log.e(LOG_TAG, "prepare() failed");
}
Log.e(TAG, "Recording Started");
```

La grabación se detiene llamando al método ParарGrabacion() desde una instancia de CasosUsoMicrófono.

La figura 23 ilustra todo el proceso llevado a cabo para realizar y almacenar la grabación.

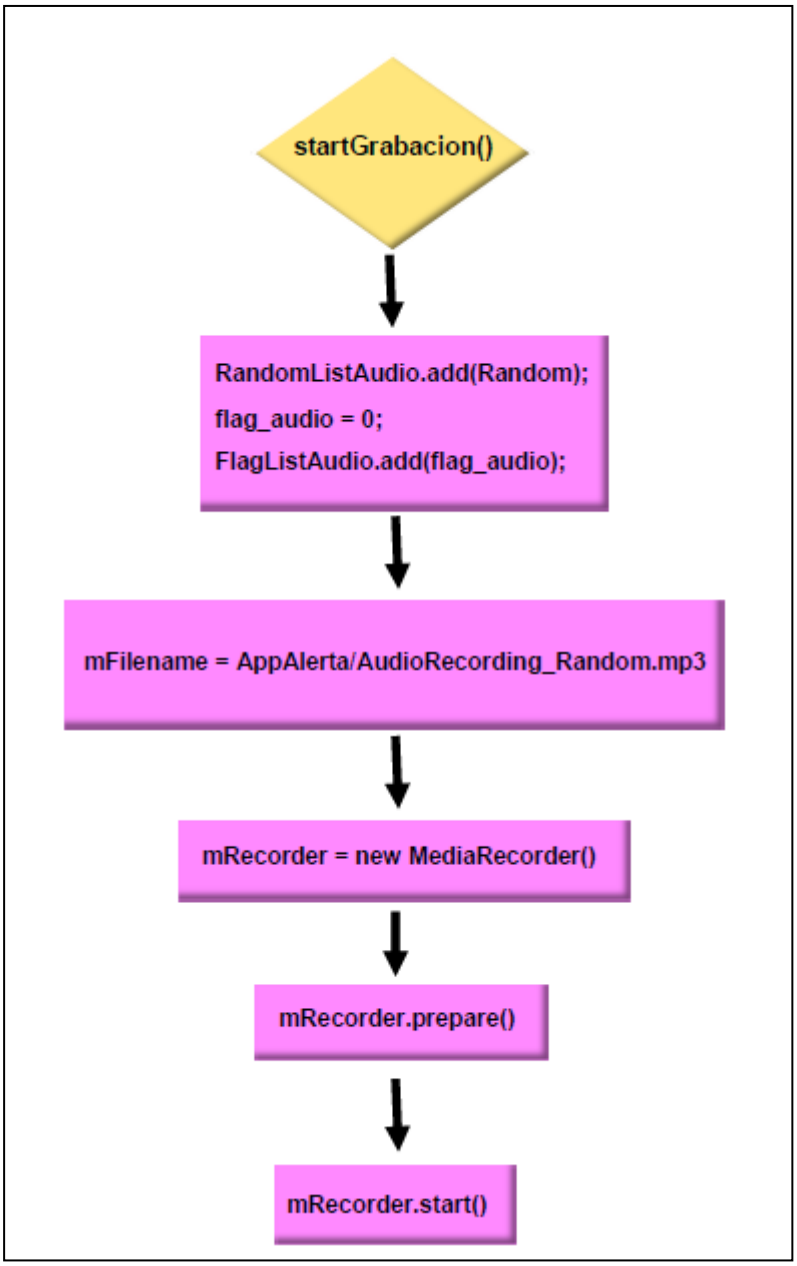


Fig. 23: Diagrama de flujo de grabación.

4.2.2.2. Capa Casos de Uso Drive

Gestiona la subida de archivos a Drive. La clase implementa ConnectionCallbacks y onConnectionFailedListener, ambas de GoogleApiClient. En su constructor recibe la actividad o contexto y un objeto de GoogleApiClient.

La subida de archivos comienza cuando, desde cualquier parte de la aplicación, se crea un objeto de tipo CasosUsoDrive y se llama al método uploadtodrive(). A continuación, se enumera el proceso de subida:

1. Uploadtodrive() llama al método CompruebaCarpeta().
2. CompruebaCarpeta() verifica la existencia de la carpeta AppAlerta en la cuenta de Drive asociada a la aplicación.

3. Si encuentra la carpeta, llama al método subir_Drive() pasando como parámetro el id de Drive para esa carpeta. Por el contrario, si no la encuentra, crea una nueva carpeta con ese nombre, y llama al método subir_Drive de la misma manera.
4. El método subir_Drive primero comprueba las preferencias y averigua qué se quiere subir a Drive (Nada, audio o localización).
   Elabora dos ramas de código con un '*switch'*, una para LOCALIZACIÓN y otra para AUDIO.

   - Localización: define el fichero PosFile con el mismo nombre con el que se guarda en el dispositivo (AppAlerta/localización.txt). Comienza a leer sobre este fichero y lo escribe en el nuevo. Genera un nuevo objeto de MetadataChangeSet al que le da título y formato.

```
final File PosFile = new
File(Environment.getExternalStorageDirectory().get
AbsolutePath() + "/AppAlerta/localizacion.txt");
try {
    FileInputStream fileInputStream = new
FileInputStream(PosFile);
    byte[] buffer = new byte[1024];
    int bytesRead;
    while ((bytesRead =
fileInputStream.read(buffer)) != -1) {
        outputStream.write(buffer, 0, bytesRead);
    }
} catch (IOException e1) {
    Log.i(TAG, "Imposible escribir fichero de
localización.");
}

changeSet = new
MetadataChangeSet.Builder().setTitle(("localizacio
n_" + String.valueOf(System.currentTimeMillis())
+".txt"
)).setMimeType("text/plain").setStarred(false).bui
ld();
```

   - Audio: De forma similar, en el caso del audio, primero busca en orden en la lista 'RandomListAudio' el número aleatorio cuyo flag siga siendo '0' , es decir, que no ha sido subido a Drive, lo toma y cambia el flag a 1.
     Con ese número aleatorio es capaz de encontrar el archivo de audio que da nombre al fichero de entrada siguiendo el mismo proceso que con el fichero de localización.

```
    for(int i=0; i<RandomListAudio.size(); i++){
        if(FlagListAudio.get(i)==0){
            FlagListAudio.set(i,1);
            Random_audio = RandomListAudio.get(i);
            break;
        }
    }
    final File audioFile = new
File(Environment.getExternalStorageDirectory().getAbsol
utePath() + "/AppAlerta/AudioRecording_" + Random_audio
+ ".mp3");

    try {
        FileInputStream fileInputStream = new
FileInputStream(audioFile);
        byte[] buffer = new byte[21000];
        int bytesRead;
        while ((bytesRead =
fileInputStream.read(buffer)) != -1) {
            outputStream.write(buffer, 0, bytesRead);
        }

    } catch (IOException e1) {
        Log.i(TAG, "Imposible escribir fichero de
audio.");
    }

    changeSet = new
MetadataChangeSet.Builder().setTitle(audioFile.getName(
)).setMimeType("audio/MP3").setStarred(true).build();
```

A continuación, la figura 24 muestra el diagrama de flujo del proceso de subida de archivos a Drive.

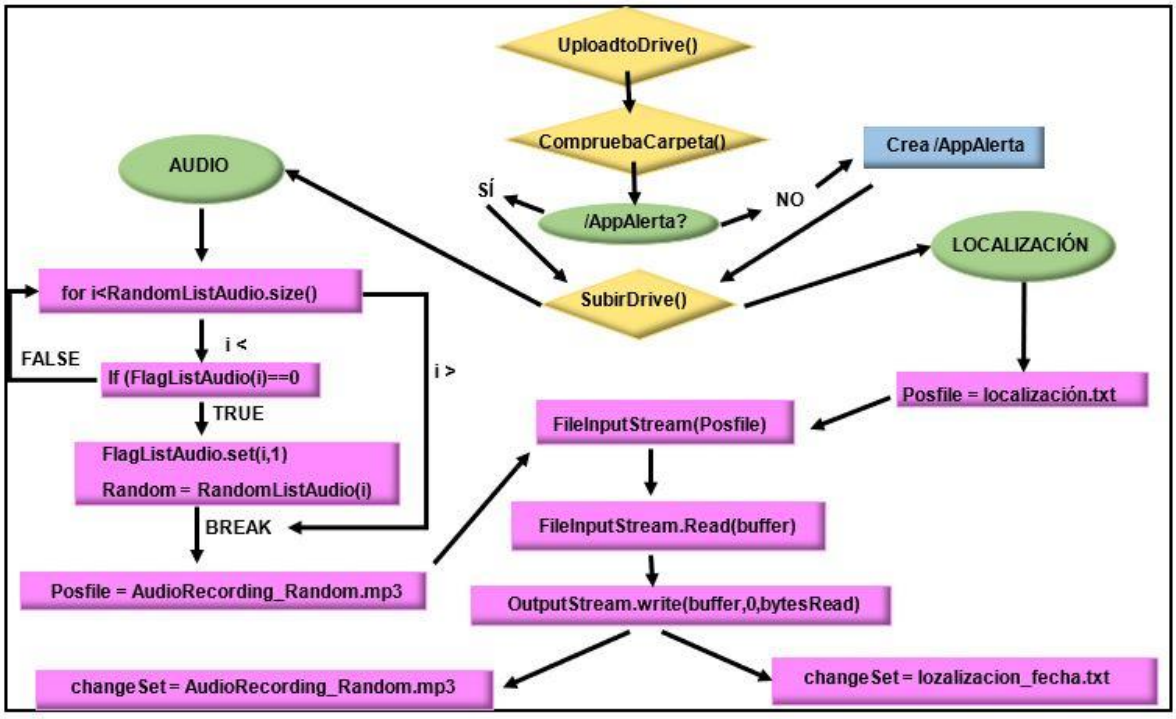


Fig. 24: Diagrama de flujo de subida a Drive

4.2.2.3. Capa Casos de Uso Localización

Esta clase se encarga de la obtención de la ubicación del dispositivo, así como de su escritura en un archivo de texto en el dispositivo. Implementa la interfaz LocationListener y recibe en su constructor la actividad y el código del permiso. Para estas acciones se hará uso de los permisos *WRITE_EXTERNAL_STORAGE y ACCESS_FINE_LOCATION*.
También en el constructor se inicializarán las variables manejadorLoc y posiciónActual. La primera, de tipo LocationManager, permite el acceso a los servicios de localización de Android. La segunda, de tipo GeoPunto, almacena la información de la mejor ubicación actual.
A continuación, se describe el proceso de obtención y escritura de localización que se sigue en esta clase.

1. En el constructor se llama al método ultimaLocalización(). Este método, tras comprobar que hay permiso de acceso a localización y almacenamiento, comprueba que los proveedores de GPS y red están habilitados y llama al método getLastKnownLocation() para cada uno de ellos.
2. Una vez obtenidos los permisos, se llama al método activarProveedores(), que maneja la actualización de los mismos. El proveedor basado en GPS se actualizará cada 10 segundos y con un cambio de más de 5 metros, el basado en redes lo hará cada 20 segundos y con un cambio de más de 10 metros.
3. Por último, en el método onLocationChanged() de la interfaz LocationListener llama al método ActualizaMejorLoc(). Este método es el que actualiza la posición guardada con la mejor localización y la guarda en la variable mejorLoc. Esta actualización se produce si la variable mejorLoc todavía no ha sido inicializada, la nueva ubicación tiene una precisión correcta (al menos la mitad de la actual), o la diferencia de tiempo es superior a dos minutos. Por últimos, la variable posiciónActual en formato GeoPunto, toma la nueva posición.

```
private void actualizaMejorLoc(Location localiz) {
    if (localiz != null && (mejorLoc == null
            || localiz.getAccuracy() < 2 *
mejorLoc.getAccuracy()
            || localiz.getTime() - mejorLoc.getTime() >
DOS_MINUTOS)) {
        Log.d(TAG, "Nueva mejor localización");
        mejorLoc = localiz;
        ((Aplicacion)
actividad.getApplication()).posicionActual.setLatitud(localiz.
getLatitude());
        ((Aplicacion)
actividad.getApplication()).posicionActual.setLongitud(localiz
.getLongitude());
    }
}
```

4. Por otro lado, también en el método onLocationChanged(), se llama al método writeToLocationFile, pasando como parámetro la localización actual.

5. El proceso de escritura es similar al del audio. Se comprueba el directorio AppAlerta, si no está creado se genera. A continuación, se crea el fichero con el nombre Localizacion.txt. En esta ocasión no será necesario añadir una secuencia aleatoria de números al nombre del archivo, ya que únicamente se creará un archivo de texto en el que se irán añadiendo mediante la opción 'append' nuevas líneas con la siguiente información:

   Longitud, latitud, altitud, precisión, hora, proveedor y velocidad.
   Se añadirá una nueva línea cada vez que el método writeToLocationFile() sea llamado. Esto ocurrirá cada vez que la localización experimente un cambio.

La figura 25 presenta el procedimiento de obtención y almacenamiento de la ubicación.

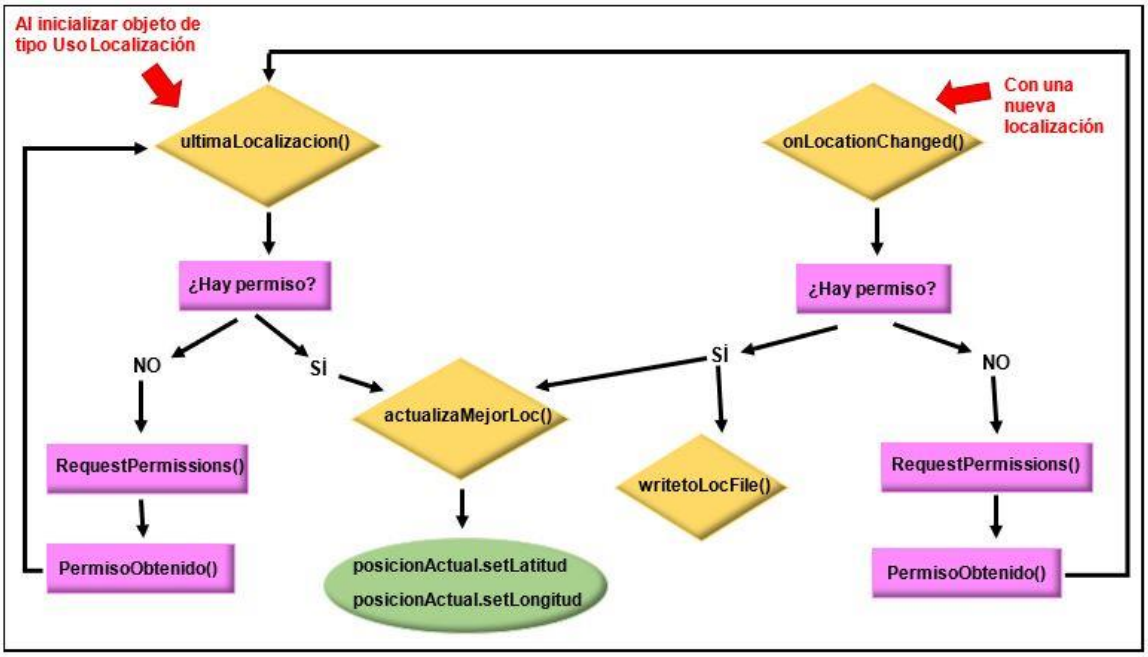


Fig. 25: Diagrama de flujo de localización.

4.2.2.4. Capa Casos de Uso Telegram

Esta clase administra los mensajes a través de Telegram. A excepción del envío del chatID del grupo de Telegram, que se hace desde otro programa, como explicaremos más adelante, el resto de las funciones se hacen desde aquí utilizando peticiones HTTP proporcionadas por la API de Telegram.

Esta clase hereda de AsyncTask<Void, Void, Void>, con lo que se trata de una actividad que funcionará en segundo plano.

Su constructor recibe la actividad y la localización en un objeto CasosUsoLocalizacion.

Lo primero que hace la clase es revisar las preferencias seleccionadas por la usuaria. Toma el ChatID que esta habrá tenido que copiar y pegar, el nombre y apellido de la usuaria, el mensaje de texto que desea enviar, y si desea enviar solo un mensaje de alerta, solo la ubicación o ambas cosas. Una vez obtenida esta información de las preferencias ya se pueden realizar las peticiones.

Se definen las dos opciones de URL que se lanzarán.

```
UrlMsj =
"https://api.telegram.org/bot%s/sendMessage?chat_id=%s&text=%s";
UrlUbic =
"https://api.telegram.org/bot%s/SendLocation?chat_id=%s&latitude=%s
&longitude=%s";
```

Y se aporta la información necesaria para según que envío.

```
if(Telegram_type.equals("Solo mensaje")){
    UrlString = String.format(UrlMsj, apiToken, ChatID, Mensaje);
}else if (Telegram_type.equals("Solo ubicación")) {
    UrlString = String.format(UrlUbic, apiToken, ChatID, latitud,
longitud);

}else if (Telegram_type.equals("Mensaje + ubicación")) {
    UrlString = String.format(UrlMsj, apiToken, ChatID, Mensaje);
    UrlString2 =  String.format(UrlUbic, apiToken, ChatID, latitud,
longitud);
}
```

Finalmente, se lanza la petición.

```
URL url = new URL(UrlString);
URLConnection conn = url.openConnection();
InputStream is = new BufferedInputStream(conn.getInputStream());

BufferedReader br = new BufferedReader(new InputStreamReader(is));
String inputLine = "";
StringBuilder sb = new StringBuilder();
while ((inputLine = br.readLine()) != null) {
    sb.append(inputLine);
}
```

4.2.3. Preferencias.

Para guardar la configuración de aplicación, AppAlerta hace uso de la clase SharedPreferences, una herramienta que permite almacenar y posteriormente consultar los datos guardados en forma de clave-valor. La clave es el identificativo o nombre que se le da al dato que se pretende guardar y el valor es el dato que puede ser de tipo *booleano, float, integer, long* o *String.*

Las preferencias de AppAlerta quedan representadas de forma visual en forma de 'formulario' a rellenar para la usuaría en el fichero info_pref.xml.

Para acceder a ellas desde cualquier parte de la aplicación es necesario crear un objeto de la clase SharedPreferences. A continuación, se iguala con el método getDefaultSharedPreferences().

```
public SharedPreferences preferencias_dueña;
preferencias_dueña =
PreferenceManager.getDefaultSharedPreferences(this);
```

Una vez hecho esto, ya se puede acceder a cualquier dato guardado en las preferencias.

En el ejemplo, la clave 'Medios' es el set que comprende los tres tipos de medios de comunicación que permite la aplicación: Llamada, SMS o Telegram. Para acceder a los elegidos por la usuaria.

```
private static set Medios =
preferencias_dueña.getStringSet("Medios",
Collections.<String>emptySet());
```

Para obtener los datos almacenados en este set se utiliza el método contains().

```
if(Medios.contains("Llamada")){

                (...)

        UsoContacto.llamarTelefono(contacto);

                (...)
}
```

4.2.4. Solicitud de permisos.

Una de las dificultades de AppAlerta es la petición de permisos. Debido a que la aplicación hace uso de múltiples permisos de tipo peligroso, el proceso de comprobación, petición y aceptación de permisos ha de hacerse de forma reiterada.

Se recuerda que los permisos que han de solicitarse son los siguientes:

- CALL_PHONE
- SEND_SMS
- READ_EXTERNAL_STORAGE
- INTERNET
- READ_CONTACTS
- WRITE_EXTERNAL_STORAGE
- ACCESS_FINE_LOCATION
- RECORD_AUDIO

La petición de estos permisos se ha de realizar en el momento en que la aplicación necesita acceder o realizar esas acciones restringidas, llamar directamente, consultar la agenda, el almacenamiento, etc.

La manera de gestionar los permisos es la siguiente. Primero, antes de realizar la acción deseada se comprueba que la aplicación ya tiene ese permiso, si lo tiene puede realizar la acción que desee, en caso contrario deberá solicitarlo llamando al método SolicitarPermiso(…) con el permiso, la justificación, el código del permiso y la actividad en la que se visualizará el mensaje.

```
if(ContextCompat.checkSelfPermission(this,
Manifest.permission.SEND_SMS)  ==  PackageManager.PERMISSION_GRANTED)
{
    servSendSMS(pos);

else{

solicitarPermiso(Manifest.permission.SEND_SMS, "Sin el permiso no se
podrá enviar SMS.", SEND_SMS, this);


}
```

El método solicitarPermiso(…), creará el cuadro de alerta con la justificación por la cual la aplicación necesita el permiso de la usuaria. Si lo acepta, el permiso será solicitado llamando al método onRequestsPermissionsResault(…) .

```
public static void solicitarPermiso(final String permiso, String
justificacion, final int requestCode, final Activity actividad) {
    if
(ActivityCompat.shouldShowRequestPermissionRationale(actividad,
permiso)) {
        new AlertDialog.Builder(actividad)
                .setTitle("Solicitud de permiso")
                .setMessage(justificacion)
                .setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int
whichButton) {
                        ActivityCompat.requestPermissions(actividad,
new String[]{permiso}, requestCode);
                    }
                })
                .show();
    } else {
        ActivityCompat.requestPermissions(actividad, new
String[]{permiso}, requestCode);
    }
}
```

El método onRequestsPermissionsResault comprueba el código del permiso para averiguar qué permiso es el que se está solicitando. Después muestra el cuadro de solicitud de permiso.

```
public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults) {

      if (requestCode == SEND_SMS) {
          if ((grantResults.length == 1) && (grantResults[0] ==
      PackageManager.PERMISSION_GRANTED)) {
              servSendSMS(pos);
          } else {
              Toast.makeText(this, "Sin el permiso, no puedo
      realizar la acción", Toast.LENGTH_SHORT).show();
          }
      }
(...)
}
```

Este procedimiento deberá realizarse por cada uno de los permisos peligrosos citados más arriba. Al pulsar el botón del pánico será necesaria la comprobación de hasta cinco permisos a la vez, o de tantos como acciones la usuaria quiera que se realicen.

4.2.5. Actividades

En Android, una actividad representa una unidad de interacción con el usuario, es decir, una pantalla de la aplicación. Dentro de la actividad se encuentran todos los métodos que tratan las interacciones con el usuario que ocurren en esa pantalla. Toda actividad ha de tener una vista asociada que representará la interfaz de usuario. Estas vistas suelen ser archivos xml de tipo *layout*. Para realizar esta asociación es necesario hacerlo dentro del método onCreate() como se muestra a continuación.

```
public class AcercaDeActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acerca_de);
    }
}
```

AppAlerta sigue una estructura Clean en cuya capa de presentación se encuentran todas las actividades de la aplicación como ya se ha mencionado.

A continuación, en la tabla 3, se enumeran todas las actividades de AppAlerta y los layouts asociados a cada una de ellas.

Tabla 3: Relación de actividades y vistas

Actividad(.java)	Vista(.xml)	Descripción
MainActivity	activity_main (layout)	Pantalla principal de AppAlerta. En ella se visualiza el botón del pánico. La actividad contiene los métodos que realizan las principales acciones de la aplicación.
ContactosActivity	contacto_individual (layout) recycler_view (RecyclerView) contactos_emergencia (layout)	Pantalla con la lista de contactos. Parte del layout de un solo contacto, y la librería RecyclerView los dispone conjuntamente en una sola pantalla que recoge el layout contactos_emergencia.

		Desde esta pantalla se puede añadir un nuevo contacto pulsando el botón '+'.
VerContactoActivity	ver_contacto (layout)	Pantalla para visualizar un solo contacto después de ser seleccionado. Desde esta pantalla se puede borrar el contacto o pasar a la pantalla de editado.
EditarContactoActivity	edición_contacto (layout)	Pantalla para adición o edición de un contacto.
InstruccionesActivity	instrucciones (layout)	Pantalla que muestra las instrucciones a seguir para poner la aplicación en marcha.
AcercaDeActivity	acerca_de (layout)	Pantalla con una explicación breve de lo que es y los objetivos de AppAlerta.
MapaActivity	mapa (layout)	Pantalla que carga el mapa de GoogleMaps con la posición del dispositivo actual.
MiInfoActivity	info_pref (PreferenceScreen)	Pantalla para la configuración de preferencias.
PanicoLanzadoActivity	panico_lanzado (layout)	Pantalla que aparece al pulsar el botón del Pánico.

Tabla 3: Relación de actividades y vistas.

4.2.6. Sincronización de actividades.

La propuesta de diseño de AppAlerta es la de una aplicación que presenta un botón del pánico capaz de desencadenar numerosas acciones en paralelo una vez pulsado.

Existen dos formas de conseguir la ejecución de tareas en segundo plano, utilizando servicios o mediante el uso de tareas asíncronas. Concretamente por medio de las clases IntentService() y de AsyncTask(). Ambas clases permiten la ejecución de un bloque de código en segundo plano, de manera asíncrona y en un hilo distinto al principal para evitar el bloqueo de la interfaz de usuario [33].

Para esta aplicación se ha decidido utilizar ambas opciones para poder experimentar y aprender su uso.

La implementación que se ha utilizado es la siguiente:

- La llamada siempre se hace como una actividad normal en primer plano. Desde MainActivity se define un objeto CasosUsoContacto y se llama de al método llamarTelefono() de esta clase.

```
UsoContacto.llamarTelefono(contacto);
```

- El envío de SMS se realiza mediante un servicio en segundo plano. Concretamente utiliza una subclase de Servicio denominada IntentService. La diferencia con la clase Service es que utiliza un subproceso de trabajo para gestionar todas las solicitudes de inicio de una en una mediante una cola de trabajo. La clase Service, por el contrario, es capaz de manejar varias solicitudes al mismo tiempo [34].

  Ya que solo necesitamos que este servicio gestione una petición de SMS cada vez que se pulsa el botón del pánico, se decide utilizar la clase IntentService(), más simple.

  Esta clase, denominada Servicio_SMS recibirá una llamada desde MainActivity al ejecutar el método servSendSMS(pos).

```
public void servSendSMS(int pos) {
    Intent serv = new Intent(this, Servicio_SMS.class);
    serv.putExtra("pos", pos);
    this.startService(serv);
}
```

  Este método iniciará el Servicio_SMS.
  El servicio será el encargado de comprobar en las preferencias si la usuaria desea enviar un único SMS a su primer contacto de la lista o si por el contrario desea enviarlo a cada uno de sus contactos. En el primer caso instanciará un objeto de tipo CasosUsoContacto y llamará al método SendSMS. En el segundo hará lo mismo por cada contacto o posición de este en la lista mediante un bucle.

```
tipo = preferencias_dueña.getString("SMS_todos", "Contacto
principal");


if (tipo.equals("Contacto principal")){
    UsoContacto.SendSMS(contacto);
}else if (tipo.equals("Todos")) {
    for (int i = 0; i < contactos.size(); i++){
        Contacto contacto_actual =
adaptador.contactoPosicion(i);
        UsoContacto.SendSMS(contacto_actual);
    }
}
```

El método SendSMS de CasosUsoContacto, gestionará la petición de envío de SMS, comprobando el tipo de SMS que se desea enviar (texto, ubicación o ubicación y texto) y realizando el envío mediante la clase SmsManager.

- Tanto la activación del micrófono y la escritura de los archivos de audio como la subida de estos a Drive se realizarán utilizando la clase AsyncTask(). Para ello se definirán dos clases Background_Drive y Background_Microfono.
  Cada una de ellas heredará la clase AsyncTask y en su método principal doInBackground() realizarán la llamada a los métodos usoDrive.upload_to_drive() para el caso de la subida de archivos a Drive, y usoMicrofono.startGrabacion() para el caso de la grabación del audio.
  La forma de ejecutar estas clases será la siguiente:

```
new Background_Drive(actividad,mGoogleApiClient).execute();
```

4.2.7. Telegram.

Como ya se ha mencionado en el punto 2.4.4., para la implementación del envío de mensajes a través de Telegram, es necesaria la utilización de una herramienta llamada bot.

Un bot permite al usuario, por ejemplo, obtener información del tiempo, noticias, recordatorios, etc., con tan solo buscar el bot específico que ofrece cada una de esas funcionalidades. Al tratarse de una API pública, los propios usuarios o desarrolladores pueden crear sus propios bots para realizar las acciones que ellos consideren. Es el caso del bot 'AppAlerta' creado para esta aplicación.

Mediante este bot la comunicación de la alerta por medio de Telegram será posible. La usuaria deberá crear un chat de grupo en Telegram al que añadirá los contactos que ella considere deban recibir sus alertas y el bot 'AppAlerta' que realizará las comunicaciones.

Para ello será necesario conocer el ID del chat del grupo. Este ID será proporcionado por el propio bot cuando la usuaria de permisos de administrador al bot y escriba '*chatId'* en el grupo.

A continuación, la usuaria deberá copiar y pegar este ID en las preferencias de su aplicación AppAlerta. De esta manera, todos los mensajes enviados desde la aplicación serán a este chat de grupo.

Para la obtención de este ID, será necesario que el bot esté ejecutándose en un servidor que recibirá las peticiones y enviará el ID para cada grupo.

El programa BotAlerta contiene la clase Main y la clase BotAppAlerta que hereda TelegramLongPollingBot.

La clase Main inicializa el bot ‘AppAlerta’.

```
TelegramBotsApi telegramBotsApi = new TelegramBotsApi();
try {
    telegramBotsApi.registerBot(new BotAppAlerta());
} catch (TelegramApiException e) {
    e.printStackTrace();
}
```

La clase BotAppAlerta contiene el método OnUpdateReceived() que comprueba las actualizaciones que ha habido para el bot. A continuación, obtiene los mensajes enviados al mismo y busca el mensaje ‘*chatID’*. Si recibe este mensaje significa que la usuaria está pidiendo el *chatID* de su grupo para copiarlo y pegarlo en su aplicación AppAlerta. Entonces el bot envía el *chatID* correspondiente a dicho grupo.

```
@Override
public void onUpdateReceived(final Update update) {
        // Se obtiene el mensaje escrito por el usuario
        final String messageTextReceived =
update.getMessage().getText();
        List<User> newUsers =
update.getMessage().getNewChatMembers();
        // Se obtiene el id de chat del usuario
        final long chatId = update.getMessage().getChatId();

        if (newUsers != null) {
            String chatIdGrupo = Long.toString(chatId);
            SendMessage chatMessage = new
SendMessage().setChatId(chatId).setText(chatIdGrupo);

            try {
                if((messageTextReceived.equals("chatid"))
||(messageTextReceived.equals("Chatid"))||(messageTextReceived.equal
s("ChatId"))||(messageTextReceived.equals("ChatID"))||(messageTextRe
ceived.equals("CHATID"))) {
                    execute(chatMessage);
                }
            } catch (TelegramApiException e) {
                e.printStackTrace();
            }
        }
}
```

4.3. Implementación de la base de datos.

La base de datos diseñada para AppAlerta está formada por la tabla 'contactos'. Esta tabla está constituida por los campos mostrados en la Tabla 4.

Tabla 4: Tabla de contactos		
Columna	Tipo de dato	Descripción
_id	INTEGER	Entero utilizado como clave principal de la tabla contactos. Introducido de automáticamente por el sistema de forma que sea único para cada contacto.
Nombre	TEXT	Nombre del contacto
Estado	TEXT	Estado del contacto: • Nuevo • En edición • Editado
Contacto_id	TEXT	Id único del contacto tomado del ContentProvider de la agenda del teléfono. Utilizado para comprobar contactos repetidos.
Fecha	BIGINT	Fecha de creación del contacto en AppAlerta.
Teléfono	BIGINT	Número de teléfono del contacto.

Tabla 4: Contactos.

La clase ContactosBD implementa la interfaz Contactos y hereda la clase SQLiteOpenHelper. Por medio de esta clase se define y gestiona esta pequeña base de datos.

La tabla se crea mediante una sencilla consulta dentro del método :

```
@Override public void onCreate(SQLiteDatabase bd) {

bd.execSQL("CREATE TABLE contactos ("+
        "_id INTEGER PRIMARY KEY AUTOINCREMENT, "+
        "nombre TEXT, " +
        "estado TEXT, " +
        "contacto_id TEXT, " +
        "fecha BIGINT, " +
        "telefono BIGINT)");
}
```

Una vez creada la tabla, ContactosBD implementa todos los métodos heredados de la interfaz Contactos en lenguaje SQL.

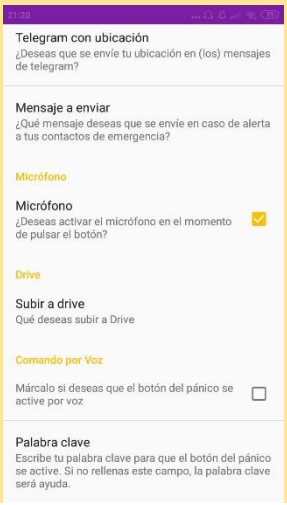
5. Resultados

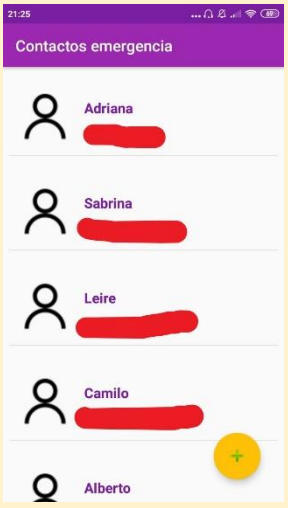
Concluido el proceso de diseño e implementación de la aplicación, se ha obtenido el prototipo final de AppAlerta.

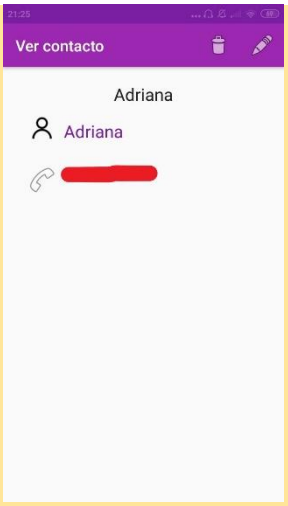
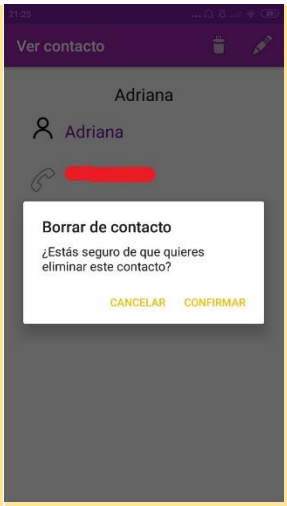
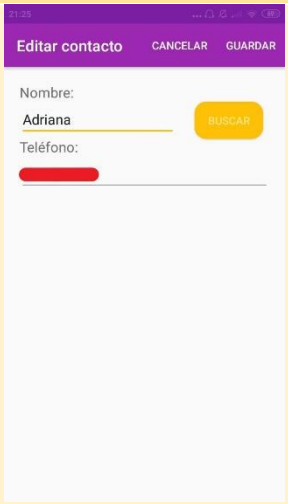
La tabla 5 muestra cada una de las pantallas de la aplicación y una breve descripción de sus características y funciones.

Tabla 5: Resultado final pantallas

Pantalla de inicio	Pantalla principal de AppAlerta. En ella se visualiza el botón del pánico. También se encuentran el botón de preferencias y el de acceso a contactos de emergencia. Se muestran algunos de los errores típicos en pantalla.	 
Pantalla de preferencias	Pantalla para la configuración de preferencias.	 

		
Pantalla 'Acerca de…'	Pantalla con una breve explicación de lo que es AppAlerta y sus objetivos.	
Pantalla de instrucciones	Pantalla que muestra las instrucciones a seguir para poner la aplicación en marcha. También incluye un tutorial para el uso con Telegram.	 

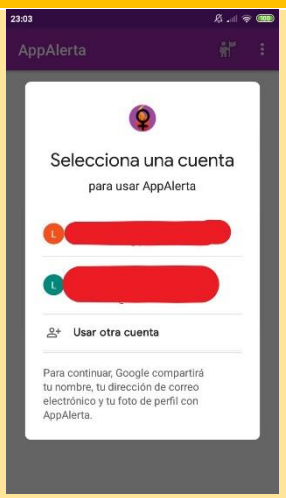
		 
Pantalla interfaz Google Maps	Pantalla que carga el mapa de GoogleMaps con la posición del dispositivo actual.	
Pantalla contactos emergencia	Pantalla con la lista de contactos. Parte del layout de un solo contacto, y la librería RecyclerView los dispone conjuntamente en una sola pantalla que recoge el layout contactos_emergencia. Desde esta pantalla se puede añadir un nuevo contacto pulsando el botón '+'.	

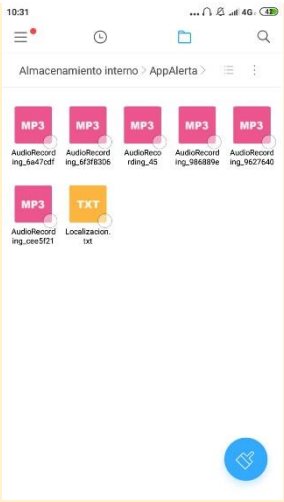
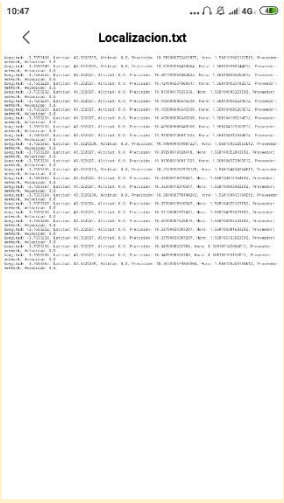
Pantalla ver contacto	Pantalla para visualizar un solo contacto después de ser seleccionado. Desde esta pantalla se puede borrar el contacto o pasar a la pantalla de editado.		
Pantalla editar-añadir contacto	Pantalla para adición o edición de un contacto. Si se quiere añadir un nuevo contacto, los campos aparecerán en blanco. Por el contrario, si se quiere editar, los campos aparecerán con los datos actuales para ese contacto. Salta un mensaje de error si se intenta guardar con los campos vacíos o si se trata de un contacto repetido.	 	 

Pantalla del pánico	Pantalla que aparece al pulsar el botón del Pánico.		

Tabla 5: Resultado final pantallas

La tabla 6 muestra los resultados finales de las comunicaciones y demás funcionalidades que se había propuesto realizase la aplicación.

Tabla 6: Resultados comunicaciones		
Log-in en Google Drive	Pantalla que se muestra en AppAlerta para seleccionar la cuenta de Drive a la que se quieren subir los archivos.	

Almacenamiento dispositivo	Archivos de audio y localización almacenados en la carpeta AppAlerta en el dispositivo.	
Telegram	Mensajes enviados al chat de grupo de Telegram por el bot ‘AppAlerta’: • Solo ubicación. • Mensaje y ubicación. • Solo mensaje.	
Archivo Localizacion.txt	Archivo .txt almacenado en dispositivo y/o en Drive con las coordenadas del dispositivo a medida que se van actualizando.	

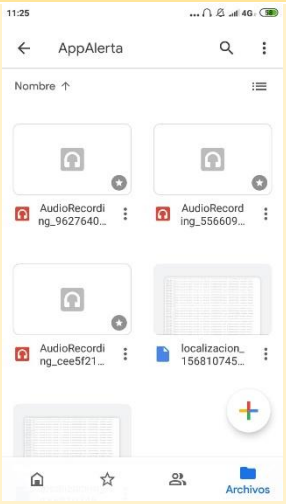
SMS	Mensajes enviados por SMS: • Solo mensaje. • Mensaje y ubicación. • Solo ubicación.	
Almacenamiento en Google Drive	Archivos de audio y localización almacenados en la carpeta AppAlerta de la cuenta seleccionada en Drive.	
Google Assistant	Pantalla que se muestra en AppAlerta si el usuario decide activar el reconocimiento de voz por medio de Google Assistant.	

Tabla 6: Resultados comunicaciones

Finalmente, en la tabla 7 se muestran las peticiones de permisos que aparecen en la pantalla.

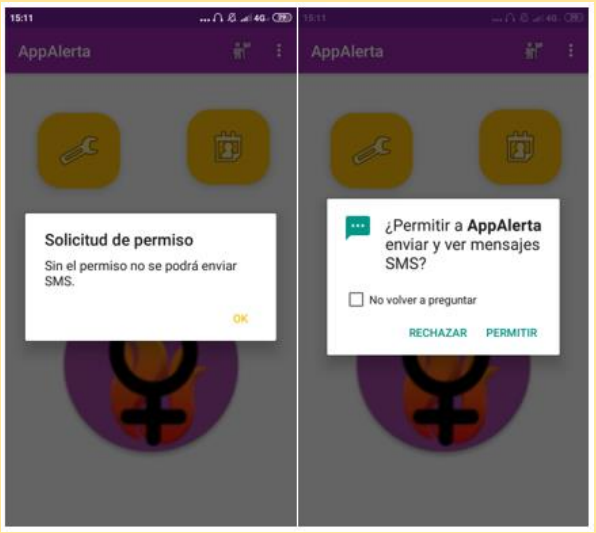
Tabla 7: Resultados permisos		
Mensajes petición de permisos.	Mensajes que se muestran de forma consecutiva a la hora de solicitar un permiso.	

Tabla 7: Resultados permisos

6. Evaluación

Con la finalidad de conocer la opinión de usuarios reales sobre esta aplicación se ha realizado un breve cuestionario a un total de 10 usuarios que han probado la aplicación y han respondido al cuestionario de forma anónima. Para obtener unos resultados más fiables sería conveniente en un futuro ampliar la muestra de usuarios.

El cuestionario realizado puede leerse en el Anexo 1 de este documento.

Obviamente, el test no se ha realizado en una situación real de peligro, con lo que los resultados más abajo analizados no serán del todo concluyentes a la hora de decidir si esta aplicación sería totalmente útil y eficiente en circunstancias de este tipo. No obstante, sí que sirve para obtener una idea aproximada del interés de potenciales clientes, y por supuesto, para conocer las mejoras que podrían aplicarse a este prototipo.

La tabla 8 muestra las puntuaciones de cada usuario.

Tabla 8: Puntuaciones evaluación

Usuario	Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4
1	6	8	8	SÍ
2	7	8	9	SÍ
3	9	9	9	SÍ
4	7	8	9	SÍ
5	6	7	8	SÍ
6	5	8	9	SÍ
7	8	8	9	SÍ
8	7	7	9	SÍ
9	7	8	9	NO
10	8	9	9	SÍ

Tabla 8: Puntuaciones evaluación.

En cuanto a la experiencia global del uso de la aplicación, la valoración ha sido en general positiva. Ocho usuarios han dado una respuesta entre el 6 y el 8, uno lo ha puntuado con un 9 y tan solo 1 le ha dado un 5 a la aplicación.

Seis usuarios han valorado la facilidad de uso de la aplicación con un 8, dos con un 9 y los otros dos usuarios restantes con un 7.

La utilidad de la aplicación ha sido la mejor puntuada con 8 y 9.

A la pregunta de si instalarían AppAlerta en sus dispositivos móviles, solo uno ha marcado la casilla 'No'.

A continuación, se visualiza un diagrama con las valoraciones (Fig. 26 y 27).

Fig. 26: Resultados evaluación 1

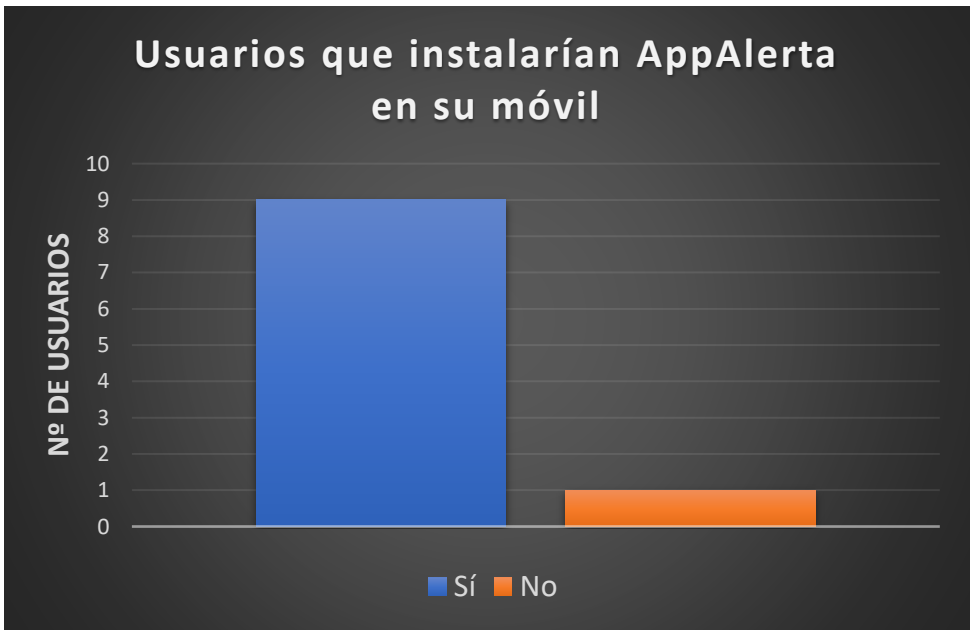


Fig. 27: Resultados evaluación 2

Por último, en referencia al apartado de la evaluación en el que se pide a los usuarios la mención de alguna mejora en la aplicación, las respuestas han sido similares. Todos han mencionado la inclusión de la aplicación de Whatsapp como servicio de mensajería en sustitución de Telegram o como alternativa a este. Algunos usuarios han propuesto el salto de llamada al siguiente contacto en la agenda si no se recibe respuesta.

7. Planificación y presupuesto

7.1. Planificación

Con el fin de ofrecer una planificación detallada del desarrollo de este proyecto, se incluye la tabla 9 en la que se enumeran las tareas llevadas a cabo para el diseño e implementación de esta aplicación. Todas ellas delimitadas por su fecha de inicio y fin y con su duración total.

Tabla 9: : Organización de tareas

	Tarea	Fecha inicio	Fecha fin	Duración (Días)
1	Comparación entre las alternativas de tecnología disponibles	03/04/2019	14/04/2019	12
2	Aprendizaje sobre la tecnología Android y la plataforma Android Studio	18/04/2019	08/05/2019	21
3	Aprendizaje básico de SQL	12/05/2019	14/05/2019	3
4	Aprendizaje sobre servicios y tareas en segundo plano	16/05/2019	21/05/2019	6
5	Investigación de aplicaciones similares	23/05/2019	25/05/2019	3
6	Recogida de requisitos funcionales de la aplicación	27/05/2019	29/05/2019	3
7	Diseño de la arquitectura en capas del proyecto	01/06/2019	03/06/2019	3
8	Diseño de la base de datos	02/06/2019	05/06/2019	4
9	Diseño de las pantallas principales	31/05/2019	07/06/2019	8
10	Implementación de la base de datos	06/06/2019	08/06/2019	3
11	Inserción de acceso a agenda de contactos	09/06/2019	10/06/2019	2
12	Implementación de preferencias	11/06/2019	12/06/2019	2
13	Obtención de localización	11/06/2019	13/06/2019	3
14	Implementación de Interfaz Google Maps	12/06/2019	13/06/2019	2

15	Implementación de SMS básico	13/06/2019	14/06/2019	2
16	Implementación de SMS con ubicación	15/06/2019	16/06/2019	2
17	Implementación de SMS a uno o más contactos	15/06/2019	16/06/2019	2
18	Implementación de llamada	17/06/2019	19/06/2019	3
19	Implementación de servicio en segundo plano para SMS	21/06/2019	23/06/2019	3
20	Implementación de micrófono	24/06/2019	27/06/2019	4
21	Implementación de almacenamiento localización.txt	25/06/2019	28/06/2019	4
22	Implementación de subida a Drive	11/07/2019	16/07/2019	6
23	Creación de tareas en segundo plano	22/07/2019	25/07/2019	4
24	Implementación de bot de Telegram	26/07/2019	29/07/2019	4
25	Sincronización de actividades y acciones	05/08/2019	09/08/2019	5
26	Implementación de pantalla de instrucciones	14/08/2019	16/08/2019	3
27	Traducción de la aplicación al inglés	17/08/2019	18/08/2019	2
28	Realización de pruebas unitarias	09/06/2019	27/08/2019	80
29	Corrección de errores	23/08/2019	27/08/2019	5
30	Realización de prueba global	28/08/2019	29/08/2019	2
31	Evaluación	06/09/2019	09/09/2019	4
32	Elaboración de memoria	22/08/2019	12/09/2019	22

Tabla 9: Organización de tareas.

Por último, se incluye un diagrama de Gantt (Fig. 28) con todas las tareas anteriormente descritas y su duración en el tiempo.

Diagrama de Gantt

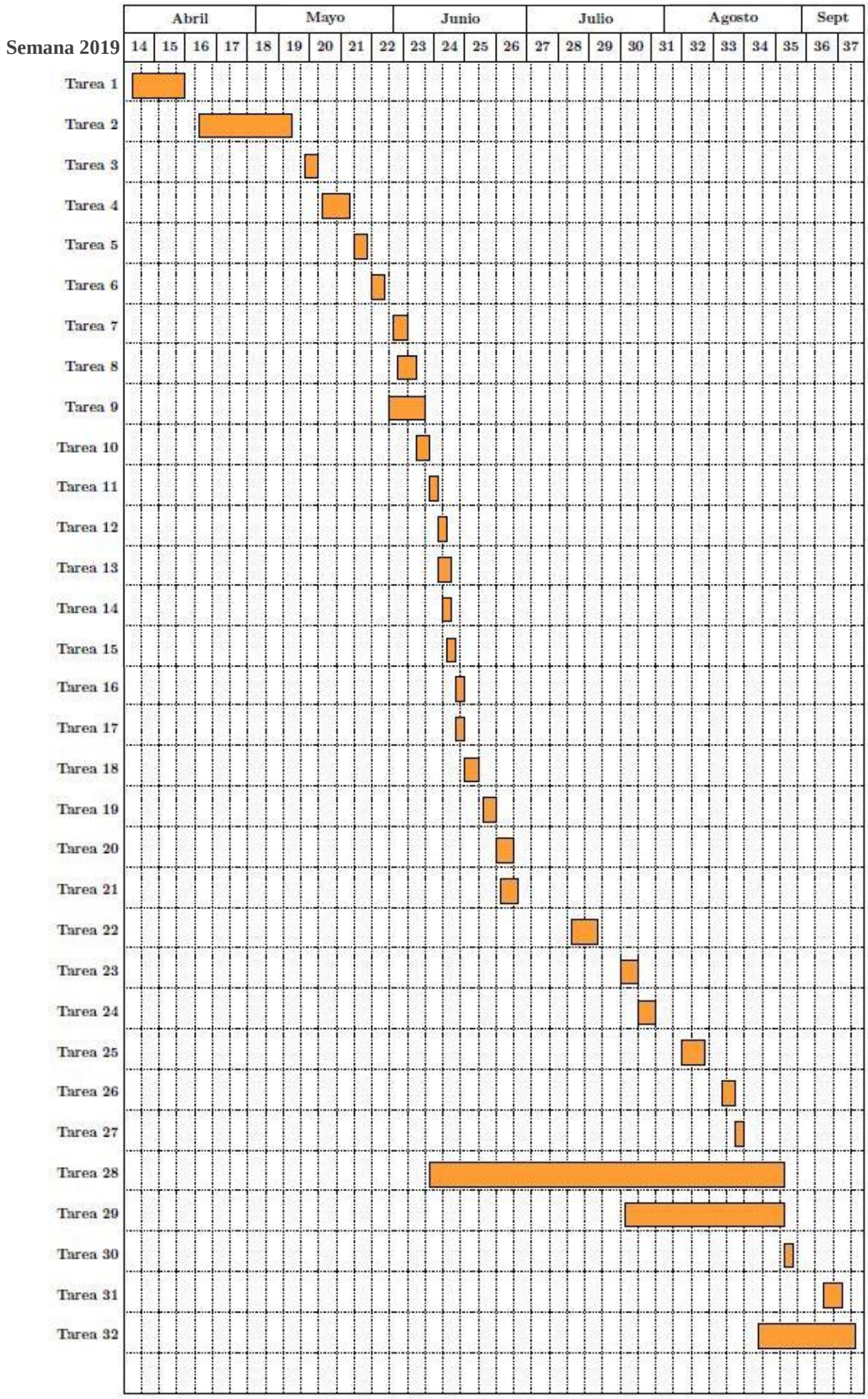


Fig. 28: Diagrama de Gantt

7.2. Presupuesto

Para hacer un cálculo aproximado del presupuesto necesario para llevar a cabo este proyecto, primero necesitamos obtener la duración del mismo. Analizando el capítulo 7.1. de planificación, teniendo en cuenta que el diseño comenzó el 3 de abril y la memoria quedará finalizada el 12 de septiembre, la duración total de este proyecto es de cinco meses y medio, 165 días.

El presupuesto deberá realizarse teniendo en cuenta tres puntos:

- **Coste de software**
  Los costes de software han sido nulos. Por un lado, el entorno de Android Studio empleado para la programación de la aplicación es gratuito. Por otro, el editor de texto utilizado para la redacción de la memoria, Microsoft Word, necesita de licencia, pero la autora disponía de una licencia gratuita como estudiante proporcionada por la Universidad Carlos III de Madrid.
- **Coste de hardware**
  Únicamente ha sido necesario el uso de un ordenador portátil para el desarrollo de la aplicación y un teléfono móvil para la realización de las simulaciones.
  El coste aproximado del portátil es de unos 600€ y el del teléfono móvil unos 130€.
  Considerando que la vida útil de un ordenador medio está entre 4 y 6 años, y la de un teléfono móvil entre 2 y 3 años, podemos hacer una estimación del coste en hardware empleado a lo largo de estos 5 meses y medio:

$$Coste\ = 600€ * \frac{165\ días}{6\ años * \frac{365 días}{año}} + 130€ * \frac{165\ días}{3\ años * \frac{365 días}{año}} = 64.79€$$

- **Personal en plantilla.**
  Las trabajadoras involucradas en la realización de este proyecto han sido la profesora en calidad de tutora Iria Manuela Estévez Ayres y la alumna Laura Fernández-Ávila Cobo estudiante de grado en Ingeniería de Telecomunicaciones.

  El sueldo por hora que debe atribuirse a cada miembro del equipo de trabajo ha de ser acorde a su rango. Para estimar el sueldo de la alumna se la considerará recién titulada a pesar de no disponer aún del título.

Tabla 10: Presupuesto en plantilla

Trabajadora	Rango	Sueldo/hora	Horas trabajadas	Total
Iria Manuela Estévez Ayres	Titulada doctora	35,33€	30	1059,90€
Laura Fernández-Ávila Cobo	Recién titulada	23,43€	540	12652,20

Tabla 10: Presupuesto plantilla.

En total los gastos atribuidos a la mano de obra serán de:

$$1059{,}90 + 12652{,}20 = 13712{,}10€$$

Por último, es cierto que podría incluirse en este presupuesto los gastos de electricidad e Internet generados para el desarrollo del mismo. No obstante, como la mayoría del trabajo se ha realizado en las instalaciones de la Universidad Carlos III que dispone de estos servicios de manera gratuita, podemos considerar estos gastos despreciables.

Con todo ello, la estimación del presupuesto total para el diseño e implementación de este proyecto es:

$$64{,}79 + 13712{,}10 = 13776{,}89€$$

8. Conclusión y líneas futuras

AppAlerta es una aplicación que pretende ser la herramienta que cualquier mujer utilizaría en una situación de alerta. Su objetivo es tener una interfaz sencilla que no lleve a distracciones o equivocaciones durante el estado de peligro. También pretende tener la mayor variedad de funciones rápidas que podrían ser de utilidad en estas circunstancias, sin pecar de características excesivas o llamativas que únicamente distraigan la atención sobre el objetivo principal: la ayuda para esa usuaria.

Finalizado el proceso de diseño e implementación, se considera que se han logrado conseguir estos objetivos.

A nivel de implementación, como se puede apreciar en el capítulo de Resultados (5.), todos los requisitos mínimos de aplicación expuestos en el diseño han sido superados.

Los resultados de la evaluación realizada en el capítulo 6. son bastante favorables. No obstante, la prueba se ha realizado sobre una muestra muy pequeña (solo 10 usuarios). Además, las pruebas se han realizado en una situación cotidiana normal, no en una de peligro como exige esta aplicación. Por todo ello, estos resultados no son del todo concluyentes, sin embargo, son de utilidad para concebir nuevas funciones y mejoras sobre esta aplicación.

En primer lugar, todos los usuarios han coincidido en la adición de WhatsApp como servicio de mensajería principal. Esta sugerencia no sorprende, puesto que como ya se ha mencionado, se trata de la aplicación de mensajería más utilizada a nivel mundial. Se considera por lo tanto una mejora casi obligatoria, aunque por desgracia, no depende del desarrollador si no de las políticas de privacidad de WhatsApp. Como se ha analizado en el punto 2.4.4., WhatsApp no dispone de una API pública, por ello las aplicaciones externas deben interaccionar con ella por medio de 'intenciones'. Las intenciones pueden lanzar acciones desde AppAlerta como el envío de un mensaje, pero WhatsApp no permite hacerlo de forma directa desde la aplicación, solo permite el acceso a su interfaz y la preparación del mensaje de texto al contacto deseado. Es el usuario quien debe pulsar la tecla de envío desde WhatsApp. Hasta que la compañía no permita realizar esta acción de forma automática desde una aplicación externa, no tendrá sentido ser añadida como medio de mensajería. Aun así, se apunta como mejora principal sobre este proyecto.

Otra mejora también señalada por alguno de los usuarios es la de implementar un sistema de llamada en bucle. A la hora de pulsar el botón de alerta se llamará al primer contacto en la lista, se esperará una serie de tonos de llamada y si no se recibe respuesta, se llamará al próximo contacto en la agenda. Esta mejora definitivamente haría del proceso de llamada una funcionalidad mucho más atractiva. Sin embargo, no es posible la realización de esta implementación. El manejador de llamadas de Android distingue entre tres estados de llamada : idle, *offhook* y *ringing*. El primero es aquel en el que no se produce ninguna actividad. El segundo hace referencia a cualquier llamada saliente que esté siendo marcada, en curso o en espera. El último caso es el de una llamada entrante [35]. Con ello, es imposible determinar de forma automática si la llamada realizada desde

AppAlerta ha sido contestada o si por el contrario sigue dando tono, ya que ambas acciones pertenecen al mismo estado (*offhook*).

Por otro lado, sería interesante el desarrollo de esta aplicación para dispositivos '*wearables'*. El uso de AppAlerta en un mecanismo de este tipo multiplicaría su utilidad. Dotaría a la usuaria de mayor independencia y rapidez de reacción, que no tendría que llevar el móvil en la mano o sacarlo aceleradamente, sino que lo haría con un gesto de muñeca. Además, Android Studio ofrece la posibilidad de desarrollo de aplicaciones para Wear OS de forma sencilla.

Finalmente, mencionar las posibilidades de escalabilidad para el servidor utilizado en el servicio de Telegram. Para la implementación de esta aplicación y las pruebas realizadas en ella, solo ha sido necesaria la ejecución del bot de Telegram en el mismo PC en el que se ha desarrollado la aplicación. No obstante, si esta aplicación saliera al mercado sería imposible que un ordenador con unas características a nivel de usuario pudiera gestionar un mínimo de peticiones. Para garantizar la escalabilidad y el rendimiento de la aplicación, se propone un despliegue en la nube que ofrece la posibilidad de contar con un servidor 'en línea', escalable y siempre disponible. Además, el capital inicial a invertir es considerablemente más bajo que el necesario para la compra de una máquina.

Bibliografía

[1] Ministerio de Sanidad, Servicios Sociales e Igualdad. "Violencia de género – datos y estadísticas". Epdata. https://www.epdata.es/datos/violencia-genero-estadisticas-ultima-victima/109/espana/106 (acceso: 25 de agosto de 2019).

[2] M. Borraz y A. Ordaz. "Las denuncias por violencia sexual han crecido un 60% en los últimos seis años". El diario. https://www.eldiario.es/sociedad/denuncias-agresion-sexual-aumentaron_0_879462206.html (acceso: 25 de agosto de 2019).

[3] Yeeply. "GDPR para apps móviles: 5 pasos para conocer las nuevas regulaciones". Yeeply. https://www.yeeply.com/blog/gdpr-apps-moviles-nuevas-regulaciones/ (acceso: 26 de agosto de 2019) .

[4] Android developers. "Permisos del sistema". Developers. https://developer.android.com/guide/topics/security/permissions.html?hl=es-419 (acceso: 26 de agosto de 2019).

[5] 112 emergencias. Comunidad de Madrid. "App de Emergencias". Madrid.org. http://www.madrid.org/112/index.php/actualidad/app-de-emergencia (acceso: 28 de agosto de 2019).

[6] S.O.S Emergencias. "El Sistema S.O.S. Emergencias". EmergenciasSOS. https://emergenciassos.com/ (acceso: 28 de agosto de 2019).

[7] T3chFest. *Bindi: tecnología para detectar y prevenir la violencia contra las mujeres | T3chFest 2019*. (enero 2019). Acceso: 28 de agosto de 2019. [Video en línea]. Disponible en: https://www.youtube.com/watch?v=M9QDgwKdzgQ

[8] Statista. https://es.statista.com/ (acceso: 28 de agosto de 2019).

[9] Statista. "Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018". https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ (acceso: 28 de agosto de 2019).

[10] Wikipedia. "Android". https://es.wikipedia.org/wiki/Android (acceso: 30 de agosto de 2019).

[11] J. Tomas Gironés. *El gran libro de Android*, 3ª edición. España: Marcombo S.A, 2013.

[12] G. Palau. "Desarrollo de apps: diferencias entre Android e iOS". Pickaso. https://pickaso.com/2019/desarrollo-apps-diferencias-android-vs-ios (acceso: 30 de agosto de 2019).

[13] Wikipedia. "iOS". https://es.wikipedia.org/wiki/IOS (acceso: 30 de agosto de 2019).

[14] D. López Villegas. “IDE: Entornos de Desarrollo Integrados para Android”. Acedemia Android. https://academiaandroid.com/ide-entornos-integrados-de-desarrollo-para-android/ (acceso: 30 de agosto de 2019).

[15] D. Gallardo. “Iniciándose en la plataforma Eclipse”. IBM. https://www.ibm.com/developerworks/ssa/library/os-ecov/index.html (acceso: 30 de agosto de 2019).

[16] A. García Pañoso. “Monitor de recursos multiplataforma con entorno de desarrollo”. DMReMon. http://dmremon.sourceforge.net/tecnologias_utilizadas_herramientas_desarrollo_es.shtml (acceso: 30 de agosto de 2019).

[17] Android developers. “Introducción a Android Studio”. Developers. https://developer.android.com/studio/intro?hl=es-419 (acceso: 30 de agosto de 2019).

[18] pedrini210. “Características y cualidades de Android Studio”. Desde Linux. https://blog.desdelinux.net/caracteristicas-y-cualidades-de-android-studio/ (acceso: 30 de agosto de 2019).

[19] Wikipedia. “NetBeans” https://es.wikipedia.org/wiki/NetBeans (acceso: 30 de agosto de 2019).

[20] NetBeans. “Información NetBeans IDE 6.1”. https://netbeans.org/community/releases/61/index_es.html (acceso: 30 de agosto de 2019).

[21] Edelete. “NetBeans”. https://www.scenebeta.com/noticia/netbeans (acceso: 30 de agosto de 2019).

[22] Universidad de Alicante. Servicio de Informática ASP.NET MVC 3 Framework. “Modelo Vista Controlador MVC”. https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html (acceso: 31 de agosto de 2019).

[23] P. Ramírez. “Modelo Vista Controlador MVC: ¿Qué es y para qué sirve?”. IT Software. https://itsoftware.com.co/content/modelo-vista-controlador-mvc-sirve/ (acceso: 31 de agosto de 2019).

[24] T. Megali. “Cómo adoptar Model View Presenter en Android”. Code tutsplus. https://code.tutsplus.com/es/tutorials/how-to-adopt-model-view-presenter-on-android--cms-26206 (acceso: 31 de agosto de 2019).

[25] Develapps. “Modelo Vista Presentador (MVP) en Android”. Develapps. http://www.develapps.com/es/noticias/modelo-vista-presentador-mvp-en-android (acceso: 31 de agosto de 2019).

[26] R.C. Martin (Uncle Bob). “The Clean Architecture”. Cleancoder. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (acceso: 31 de agosto de 2019).

[27] L. Vespa. “Patrones Arquitectónicos en Android”. Medium. https://medium.com/@vespasoft/patrones-arquitect%C3%B3nicos-en-android-ded39f7a2c10 (acceso: 31 de agosto de 2019).

[28] Whatsapp “Features”. Whatsapp. https://www.whatsapp.com/features/ (acceso: 31 de agosto de 2019).

[29] G. Torresi. “Whatsapp supera los mil millones de usuarios diarios activos”. La Vanguardia. https://www.lavanguardia.com/tecnologia/20180202/44448498399/whatsapp-usuarios-facebook-mark-zuckerberg.html (acceso: 31 de agosto de 2019).

[30] Telegram. “FAQ”. Telegram. https://telegram.org/faq (acceso: 31 de agosto de 2019).

[31] T. Castillo. “Telegram anuncia 200 millones de usuarios mensuales activos y presume de privacidad”. Genbeta. https://www.genbeta.com/mensajeria-instantanea/telegram-anuncia-200-millones-de-usuarios-mensuales-activos-y-presume-de-privacidad (acceso: 31 de agosto de 2019).

[32] E. Marín. “Chat es la nueva app de mensajería de Google para Android con la que quieren combatir a WhatsApp”. Gizmodo. https://es.gizmodo.com/chat-es-la-nueva-app-de-mensajeria-de-google-para-andro-1825415600 (acceso: 31 de agosto de 2019).

[33] K. Hoffman. “Using IntentService vs AsyncTask in Android”. AndroidPub. https://android.jlelse.eu/using-intentservice-vs-asynctask-in-android-2fec1b853ff4 (acceso: 2 de septiembre de 2019).

[34] Android developers. “Descripción general de los servcios”. Developers. https://developer.android.com/guide/components/services (acceso: 2 de septiembre de 2019).

[35] Android developers. “Telephony Manager”. Developers. https://developer.android.com/reference/android/telephony/TelephonyManager.html (acceso: 10 de septiembre de 2019).

Abstract

Introduction

It has been designed and implemented a mobile application for women called 'AppAlerta'. The aim of this application is to serve as an emergency tool with which the user will be able to ask for help if she is in a dangerous situation of any kind.

The application allows the user to communicate her situation by three different kinds of media: telephone call, SMS or Telegram text message. In addition, it is also able to record audio, access the telephone location and upload information into a Drive account in order to somehow register everything happening around the dangerous situation.

Therefore, the goal of AppAlerta is to help the woman in danger to communicate her status in the fastest and safest way possible and receive help. However, it also expects to gather as much information as possible that could serve as evidence in a trial.

This application has been developed in Android Studio, a simple environment to programme Android applications.

It was required to have some knowledge in Java, and skills where gained related with Android technology. It was also necessary to develop a small data base in SQLite, as well as a bot in Telegram.

Finally, the application makes use of several Google APIs such as Drive's, Maps' and Google Assistant and also Telegram's API.

Objectives

The goal is to obtain an easy application that will allow the woman to quickly ask for help just by pressing one single button in the middle of the screen. This button will be able to perform a series of actions such as a phone call, SMS, audio recording, etc., all at once, depending on the actions preconfigured by the user earlier. The application will also provide a small list of 'emergency contacts' defined by the user and will contact them by SMS or phone call.

In order to be able to develop such an application, several objectives have been settled:

- Selection of the appropriate programming language.
- Decision of the correct environment to develop such language.
- Learning of this language, the environment or both.
- Development of a small database to keep the emergency contacts introduced by the user.
- Research and implementation of background activities.
- Autonomy and control of the applications towards the different actions so that the user will only need one click to perform al lot them at once.
- Implementation of a bot in Telegram.
- Learning and use of Google's and Telegram's APIs.

State of art

In order to develop this project a big amount of research was done in relation with different technologies that could be applied to it. Four main points were considered:

Previous work

Some applications were studied and compared in order to gain some knowledge of what is available currently in the market. Three interesting technologies were compared:

- **My112**: A mobile application created in the Community of Madrid that performs in case of emergency a phone call with location to the 112. It also allows the creation of a personal contact list to whom send and SMS after the 112 calling. Finally, it is also possible to subscribe to the different integrated centres to receive alerts in real time of the emergencies happening in your urea.
- **S.O.S. Emergencias:** This is another mobile application which belongs to a greater system designed by a firefighter's team. This application allows citizens to perform an emergency call when in danger. However, it only allows to send the location when the call is done to service members.
- **Bindi:** It is a device that is being designed and developed by UC3M4Safety team since 2016. Its aim is to detect and prevent violence against women. The device consists of adding sensors to the famous wearables in order to detect the victim's emotions and measure the hearing environment with the idea to decide if a dangerous situation is taking place.

First two technologies are similar to what wants to be created, a mobile application that performs an alert when pressing a button. However, AppAlerta wants to be more complete and introduce more options to perform this alert, not only calling but sending SMS or Telegram Message. Also, AppAlerta wants it to happen in an automatic way, the call will directly be done when pressing the button, not by pressing twice as done in the two first technologies. Also, it is possible to introduce as an emergency contact 112 number or any other official emergency number, however it will not be able to send it its location since it does not have any agreement with the Emergency Centre.

Application

At the application level, different technologies were compared and chosen to design this project.

- **Mobile Platform.**
  The two main technologies were described and compared, Android and iOS. Android having the largest market rate was the one chosen. The aim of this application is to reach the most users as possible. Also, Java is the main language used to program Android, and the developer already knows it. The developer and designer of this application owns an Android mobile phone; therefore it makes it easier when having to test the application. These and the adaptability of Android to any kind of hardware are the main reasons why this technology was chosen.

- **Integrated Development Environment (IDE)**
  The main environments normally used to develop applications in Android are Eclipse, Android Studio or NetBeans. This project was carried out using Android Studio. This technology is today the official environment to develop Android Mobile Applications, therefore, although the other two are as useful, Android Studio is focused on these types of applications.
- **Architectures**
  It has been decided to use some kind of architecture to organize the code and its classes and have some better structure of the project and its functionalities. Three main architectures were considered: MVC (Model – View - Controller), MVP (Model – View - Presenter) and Clean architecture.
  The first two are very similar. Model layer which includes all the application data, and view layer which shows this data to the user. The last layer, Controller and Presenter respectively, both are used as intermediary between Model and View layers. The main difference between these two architectures is that in MVP, the View receives no information from the Model layer, therefore there is a greater Independence between the layers.
  Clean architecture is formed by four layers: Model, Data, Use Cases and Presentation layers. The first one includes the classes which represent the inner structure of the project. The second one the data classes and the third one includes the classes defining the different actions which the user will be able to do when using the application. Finally, the last one will include the classes that interact directly with the user such as activities or fragments.
  Since the actions of this project are very well defined (call, send SMS, record…), it was considered that the Clean Architecture would offer a better and more clear view of the code and its structure.

Messenger services

Apart from the traditional phone call and the SMS, AppAlerta wants to provide some extra media to send an emergency message using the Internet. Three main Messenger service technologies are compared: WhatsApp, Telegram and Google Chat.

Google Chat is quickly discarded since it has not a big acceptance by users yet and does not provide many new functionalities apart from the ones in SMS. WhatsApp is the most used application to send and receive messages all over the world. However, it has an inconvenient, it does not provide a public API for developers, therefore the interaction between an external application and WhatsApp can only be done by using Intents. With intents it is possible to send a message to a specific user through WhatsApp, but it will not let the application activate the ‘send’ button from WhatsApp, the user will have to do it. This inconvenient is the reason why WhatsApp is discarded and Telegram is used instead. Telegram has an available public API, and allows with the creation of a bot, to send automatic notifications to chat groups.

Storage

Android offers several ways to store information from an application. For AppAlerta some of these options have been used.

- SQLite database to store the emergency contact list.
- Preferences for the panic button configuration and rest of main settings.
- Mobile Files to store the audio and location files.
- Google Drive as an extra option to upload the audio o location files.

Design

The following requirements were defined during the design of this application:

- **Basic functionalities**
    1. Preferences configuration.
    2. Contact list creation and management.
    3. Access to contact agenda from the mobile phone.
    4. Screens design.
    5. Permission requests.
    6. Errors administration.
    7. Activities synchronization.
    8. Implementation of all actions at once when pressing the panic button.
    9. English version.
- **Communication**
    1. Automatic phone call to a contact from the list
    2. Automatic SMS with predefined text to a contact from the list.
    3. Automatic SMS with predefined text to all contacts from the list.
    4. Automatic SMS with location to a contact from the list.
    5. Automatic SMS with location to all contacts from the list.
    6. Automatic SMS with predefined text and location to a contact from the list.
    7. Automatic SMS with predefined text and location to all contacts from the list.
    8. Automatic Telegram message with predefined text, location or both to group chat.
- **Location**
    1. Access to telephone location.
    2. Writing location coordinates in a text file and its storage in the mobile device.
    3. Writing every ‘x seconds’ to pretend from losing information.
- **Microphone**
    1. Microphone activation.
    2. Storing audio files in telephone memory.

3. Writing audio files every ‘x seconds’ in order to avoid the loss of information.

- **Drive**
  1. Access to the Drive user’s account.
  2. Uploading of text files (location) to Drive.
  3. Uploading of audio files (microphone) to Drive.
- **Google Assistant**
  1. Access to Google Assistant.
  2. Enabling Panic Button with Google Assistant when key word is recognized.

All these requirements need to be fulfilled in order to develop AppAlerta application. Main points to take care of are the permission requests for every action needed, the preferences configuration that the user will need to pre-set and the main activities (screens) with which the user will interact.

The screens from AppAlerta will be the following:

- **Main screen:** The principal buttons are shown in this screen. Preferences button in order to set the Panic Button configuration. Contacts button to access the emergency contacts management. The Panic Button itself and some menu bar for instructions and Google Maps.
- **Emergency Contacts screen:** All contacts are displayed in a list. Each of them can be selected by pressing on it. A plus button is at the bottom of the screen if new contacts want to be added.
- **‘See contact’ screen:** Information o fan specific contact is displayed. Delete and Edit options are shown in the upper menu.
- **‘Edit contact’ screen:** This screen allows the user to edit or create a new contact. It can be selected from the contact phone agenda.

Implementation

Used media

In order to develop this project, both the mobile application and the Telegram bot, a laptop with the following characteristics was used:

- Intel Core i7 6500U processor, 2.59 GHz.
- 8GB of memory.
- SSD hard disk of 250 GB.
- Operative System: Windows 10 Home.

The integrated development system used has been Android Studio 3.1.

In order to fulfil this document Microsoft Word 2016 has been used as text editor.

Clean architecture.

The project was organized using Clean Architecture. The arrangement of classes was done as it is shown in figure 29.

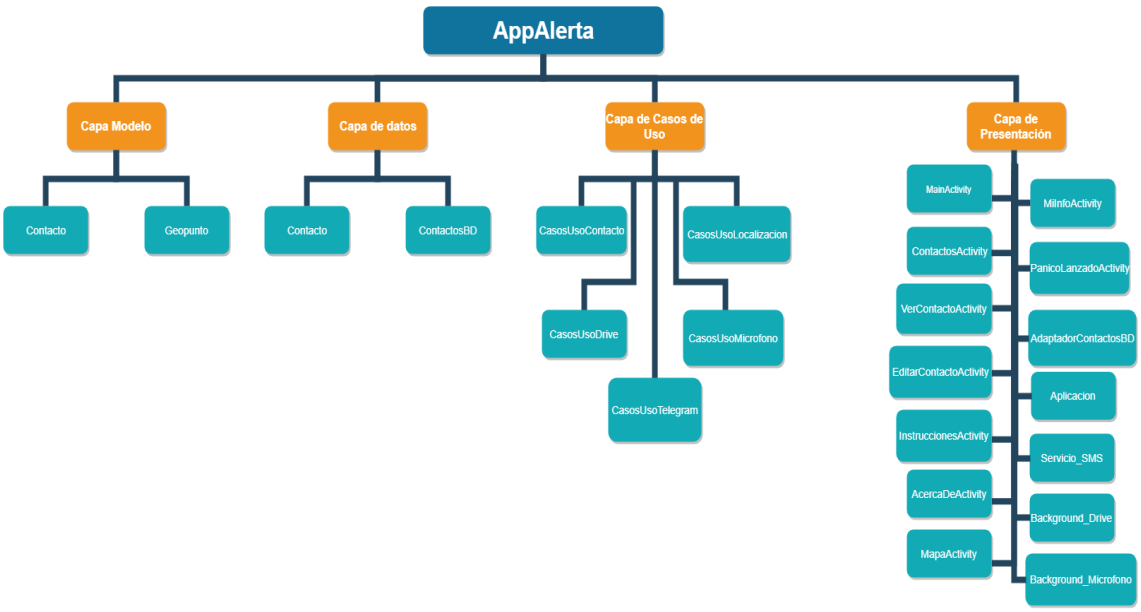


Fig. 29: AppAlerta architecture.

The classes conforming each layer and its utility is described below:

- **Model layer**
    3. **Contacto:** Class which defines the object Contact.
    4. **GeoPunto:** Class that defines the object GeoPunto, used to get the location of the mobile phone.
- **Data layer**
    3. **Contactos:** Interface which defines the different operations available on a contact list.
    4. **ContactosBD:** Class which creates and administrates the contact database.
- **Use Case layer**
    6. **CasosUsoContacto:** Class which contains all actions done on the Contact object.
    7. **CasosUsoDrive:** Class which implements Drive's APIs and performs the methods in order to upload files to Drive.
    8. **CasosUsoLocalizacion:** Class which carries out the actions in order to obtain and manage de device's location.
    9. **CasosUsoMicrofono:** Class that implements all methods related to the microphone enabling and its storage in a file in the mobile phone.
    10. **CasosUsoTelegram:** Class that manages Telegram messages.

- **Presentation layer**
    17. **MainActivity:** Main screen from AppAlerta in which the principal application's actions take place.
    18. **ContactosActivity:** Contact list screen.
    19. **VerContactoActivity:** Screen that displays one single contact.
    20. **EditarContactoActivity:** Addition or edition contact screen.
    21. **InstruccionesActivity:** Instructions screen.
    22. **AcercaDeActivity:** Short explanation screen.
    23. **MapaActivity:** Screen which loads Google Maps map with current location.
    24. **MiInfoActivity:** Preferences configuration screen.
    25. **PanicoLanzadoActivity:** Screen displayed after pressing the Panic Button.
    26. **AdaptadorContactosBD:** Class which implements ContactosActivity using the RecyclerView.
    27. **Aplicación:** Generic class that allows the access to objects belonging to any part of the project.
    28. **Servicio_SMS:** Background service that implements the SMS message sending.
    29. **Background_Drive:** Background activity that executes the actions from CasosUsoDrive.
    30. **Background_Microfono:** Background activity that executes the actions from CasosUsoMicrofono.
    31. **DriveRepeat:** Class that inherits TimerTask in order to execute repeatedly Drive actions.
    32. **MicroRepeat:** Class that inherits TimerTask in order to execute repeatedly recording actions.

Results and evaluation

The final prototype of AppAlerta was obtained with the designed screens and functionalities. A group of 10 users tried the application and fulfilled a test with a series of questions evaluating the application. Although the sample is small and the users were not experiencing a panic situation, the results have been taken in consideration, knowing that a better and broader test needs to be done.

The evaluation was in general satisfactory. Most users considered AppAlerta a useful and easy to use application. Al lot them but one, acknowledge that they would install i ton their telephones. When asking for improvements, al lot them agreed in changing or adding WhatsApp as texting media.

Planification and Budget

The work needed to be done for this project was splitted up into a total of 32 tasks. Each of them was assigned to a date and a duration. In total, the project estimation time was of 165 days.

The Budget associated to this project was calculated taking in account the software and hardware costs, as well as the labour costs. In total, the estimated budget is **13776.89€.**

Conclusions and future work

As observed in the results and evaluation, the general performance of the application is satisfactory. All main designed requirements were fulfilled, and the desired actions are done by AppAlerta. However, there are still improvements that could be done to the application:

- The addition of WhatsApp as a messaging media. As explained before this implementation will not be able to be added until WhatsApp privacy policies are changed.
- The implementation of a 'loop calling' so that the application calls the next number in the contact list after a few tones. This implementation does not depend on AppAlerta but on the Android Calling Manager. Until it is possible to differ a call-in progress from an outgoing call on hold, this will not be able to be done.
- Implementation of AppAlerta for wearables.
- Upgrading to a server in the cloud to guarantee scalability and availability.

Anexos

Anexo 1: Encuesta de experiencia de usuario de AppAlerta

uc3m | Universidad Carlos III de Madrid

Encuesta sobre experiencia de usuario de la aplicación AppAlerta

1. Valora la experiencia global del uso de AppAlerta:

	1	2	3	4	5	6	7	8	9	10	
Muy deficiente	○	○	○	○	○	○	○	○	○	○	Muy satisfactoria

2. Puntúa la facilidad de uso de la aplicación:

	1	2	3	4	5	6	7	8	9	10	
Muy poco intuitiva	○	○	○	○	○	○	○	○	○	○	Muy intuitiva

3. Valora la utilidad de la aplicación:

	1	2	3	4	5	6	7	8	9	10	
Ninguna utilidad	○	○	○	○	○	○	○	○	○	○	Muy útil

4. ¿Instalarías esta aplicación en tu dispositivo móvil?

Sí: ◯ **No:** ◯

5. ¿Qué mejoras añadirías a la aplicación?